Inhaltsverzeichnis

Abbildungsverzeichnis I
Abkürzungsverzeichnis III
Danksagung V
1 Einleitung 1
1.1 Thema der Diplomarbeit 1
1.2 Aufgabenstellung 1
1.3 Motivation 1
1.4 Methodische Vorgehensweise 2
2 Allgemeine Grundlagen 3
2.1 Vorgehensmodelle zur Softwareentwicklung 3
2.1.1 Wasserfallmodell 5
2.1.2 V-Modell 7
2.1.3 Prototypen-Modell 8
2.1.4 Agile Softwareentwicklung 8
2.1.5 Extreme Programmierung (XP) 10
2.1.6 Fazit 10
2.2 Testfallentwurfsverfahren 10
2.2.1 Whiteboxtest und Blackboxtest 10
2.2.1.1 Whiteboxtest 10
2.2.1.2 Blackboxtest 11
2.2.1.3 Fazit zu Whiteboxtsts und Blackboxtests 11
2.2.2 Funktionale und nichtfunktionale Tests 12
2.2.2.1 Funktionale Tests 12
2.2.2.2 Nichtfunktionale Tests 12
2.2.3 Testtechniken 13
2.2.3.1 Überdeckungstests (Codecoveragetests) 13
2.2.3.2 Unittest Integrationstest 14
2.2.3.3 Protokolltest 15

2.2.3.4 Funktionaltest 15
2.2.3.5 Kompatibilitätstest 15
2.2.3.6 Performancetest (Leistungstest) 16
2.3 Testwerkzeuge 16
2.3.1 Whiteboxtestwerkzeuge 17
2.3.1.1 Debugger 17
2.3.1.2 Profiler 17
2.3.1.3 Coverage 17
2.3.1.4 Unitframework 17
2.3.2 Blackboxtestwerkzeuge 18
2.3.2.1 TTCN 18
2.3.2.2 IPS 18
2.4 Sprachübertragung durch VoIP 21
3 Funktionaltest 25
3.1 Allgemeines 25
3.2 Testaufbau 25
3.3 Testvorbereitung 26
3.3.1 Die Cotrol-Datei 27
3.3.2 Die Script-Datei 28
3.4 Testdurchführung 28
3.5 Auswertung 29
4 Praktikumsversuch zum Thema Testen 31
4.1 Einleitung zum Versuch 31
4.2 Grundlagen 32
4.3 1. Versuchsaufgabe: Registrierung 37
4.4 2. Versuchsaufgabe: Verbindungsaufbau zu einem SIP-Telefon 46
4.5 3. Versuchsaufgabe: Verbindung zu einem ISDN-Telefon 50
4.6 Anhang zum Versuch 51
4.6.1 Nachrichten 51
4.6.1.1 Request-Nachrichten 51
4.6.1.2 Response-Nachrichten 51
4.6.2 Trace eines erfolgreichen Tests 52

5 Zusammenfassung und Schlussfolgerungen 59
6 Anlagen 61
6.1 Script-Datei des Funktionaltests Kapitel 3 61
6.2 Quelltext der Scrip-Datei des Praktikumsversuches 63
Literaturverzeichnis 67
Erklärung 71

Abbildungsverzeichnis

Abbildung 1: Einordnung von Prozessmodellen [Liggesmeyer, 2009] 4
Abbildung 2: Wasserfallmodell 6
Abbildung 3: V-Modell 7
Abbildung 4: IPS-Fenster 19
Abbildung 5: IPS Architektur [Nokia-Siemens-Networks, 2009] 20
Abbildung 6: Request- und Response-Nachricht 22
Abbildung 7: Versuchsanordnung 25
Abbildung 8: Sequenzdiagramm Funktionaltest 27
Abbildung 9: Quelltext der Conrtol-Datei 28
Abbildung 10: Versuchsaufbau 32
Abbildung 11: Wasserfallmodell 33
Abbildung 12: Sequenzdiagramm Registrierung 38
Abbildung 13: Request URI 39
Abbildung 14: Insert Parameter 39
Abbildung 15: Contact 39
Abbildung 16: Insert Parameter 40
Abbildung 17: To 40
Abbildung 18: Insert Parameter 40
Abbildung 19: From 41
Abbildung 20: Register-Header 41
Abbildung 21: Messagetype (100) 42
Abbildung 22: Messagetyp (401) 42
Abbildung 23: Messagetyp (100) 43
Abbildung 24: Messagetyp (200) 43
Abbildung 25: Request URI 44
Abbildung 26: Speichern 44
Abbildung 27: Kompilieren 45
Abbildung 28: Teststart 45
Abbildung 29: Sequenzdiagramm Verbindung zum SIP-Telefon 46
Abbildung 30: Request URI 47
Abbildung 31: Messagetyp (407) 47
Abbildung 32: Nachricht auswählen 48
Abbildung 33: Messagetyp (180) 49

Abbildung 34: Teststart.....50
Abbildung 35: Rufnummer ISDN-Telefon 50

Abkürzungsverzeichnis

ACK	Acknowledge
ENUM	TElephone Number URI Mapping
ETSI	European Telecommunications Standards Institute
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPS	Independent Protocol Simulator
ITU-T	International Telecommunication Union, T steht für Endgeräte und Protokolle
RFC	Request for Comments
RTP	Realtime Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TTCN	Tree and Tabular Combined Notation
TTCN 3	Testing and Test Control Notation
URI	Uniform Resource Identifier
VoIP	Voice over IP

Danksagung

Allen, die mir mit kritischen Anregungen und fachlichem Rat bei der Anfertigung der vorliegenden Arbeit zur Seite standen, gilt an dieser Stelle mein Dank. Im Besonderen danke ich Herrn Prof. Dr. rer. nat. Sergej Alekseev und Herrn M. Sc. Rico Thomanek für die kritische Durchsicht des Manuskriptes und für die Beantwortung von Fragen bezüglich des IPS-Programms und der Erstellung des Paktikumsversuches.

1 Einleitung

1.1 Thema der Diplomarbeit

Die Diplomarbeit befasst sich mit dem Testen von Kommunikationssystemen. In diesem Zusammenhang werden Vorgehensmodelle zur Softwareentwicklung und Testfallentwurfsverfahren betrachtet. Die rasante Entwicklung immer neuer Systeme im Bereich der Kommunikation macht es erforderlich, diese ihren Anforderungen entsprechend zu testen. Eine Auswahl an Möglichkeiten und Methoden zum Testen soll beschrieben werden. Das Testen hat schon in der Vergangenheit immer eine Rolle gespielt und wird auch in der Zukunft an Bedeutung zunehmen.

1.2 Aufgabenstellung

Es soll ein Praktikumsversuch für die Studenten der Informations- und Elektrotechnik erstellt werden. Der Schwerpunkt des Praktikumsversuchs ist die Phase des Testens. Es soll die Einteilung der Testtechniken in Whitebox- bzw. Blackboxtest und Funktionale bzw. Nichtfunktionale Tests und dazugehörende Testwerkzeuge erläutert werden.
Im theoretischen Teil des Praktikumsversuches soll der Entwicklungsprozess im Bezug auf Produkte der Kommunikationstechnik beschrieben werden. Um den Entwicklungsprozess zu vereinfachen, gibt es Vorgehensmodelle. Die Vorgehensmodelle Wasserfallmodell, V-Modell, Agile Softwareentwicklung und extreme Programmierung sollen erläutert werden. Die einzelnen Entwicklungsphasen für den Entwicklungsprozess werden beschrieben.
Im praktischen Teil des Versuchs sollen ausgewählte Funktionalitäten eines Kommunikationssystems mit Hilfe einer Software untersucht werden. Als Software soll IPS von der Firma Nokia Siemens Networks zur Verwendung kommen.

1.3 Motivation

Das Testen von modernen Kommunikationssystemen, insbesondere das Testen der Software für Kommunikationssysteme, ist zu einer sehr allgemeinen und umfangreichen Angelegenheit geworden. Eine besondere Herausforderung dabei ist, Fehler in den Kommunikationssystemen zu erkennen, die nur unter seltenen Bedingungen auftreten. Es muss ein ausgewogenes Verhältnis zwischen dem Testaufwand und dem Nutzen bestehen. Bei sehr komplexen Kommunikationssystemen können Tests, welche alle möglichen Fälle untersuchen, mit zu hohen Kosten verbunden sein. Dennoch sind Tests ein wichtiger Bestandteil bei

der Einführung neuer Kommunikationssysteme. Je sensibler der Anwendungsbereich des einzelnen Systems ist, desto umfangreicher müssen die Tests für das System sein. Im Vergleich der Aufwendungen für die Entwicklung und die Tests ist davon auszugehen, dass der geringere Teil der Kosten für die Entwicklung und der größere Teil für anschließende Tests aufgebracht werden muss. Um zu verdeutlichen, was geschehen kann, wenn Störungen in der Software von Kommunikationssystemen auftreten, soll ein Störungsfall wiedergegeben werden, der sich am Dienstag den 21.April 2009 bei T-Mobile ereignet hat. „Am Dienstagnachmittag etwa ab 15.45 Uhr hatten T-Mobile-Kunden mit ihren Handys nicht mehr telefonieren können. Ursache für die Störung war nach Angaben der Telekom-Tochter ein Softwarefehler im sogenannten Home Location Register (HLR). Dieses stellt die Verbindungen zwischen den Mobilfunkstationen und den SIM-Karten und damit den Handy-Nummern her. Insgesamt drei Datenbanken seien betroffen gewesen.“ [ddp.djn/jwu/pon] Dieser Fall macht deutlich, wie wichtig das Testen ist. T-Mobile hat in Deutschland ca. 40 Millionen Kunden. Wenn da das gesamte Netz ausfällt, hat das schwerwiegende Folgen.

1.4 Methodische Vorgehensweise

Die Arbeit gliedert sich in nachfolgende drei Teile:

- Grundlagen
- Eigenversuch
- Praktikumsversuch

Die Grundlagen dienen dem inhaltlichen Gesamtverständnis der Arbeit zur Realisierung der Aufgabenstellung aus der Sicht bestehender Zusammenhänge und von Lösungsansätzen. Sie umfassen Ausführungen zu Vorgehensmodellen zur Softwareentwicklung, Testfallentwurfsverfahren und zu Testwerkzeugen. Mit den Ausführungen zu VoIP und SIP werden Wege der Sprachübertragung über Datennetzwerke aufgezeigt.
Mit dem Eigenversuch (Funktionaltest) werden der erarbeitete Testaufbau und die Testvorbereitung sowie die Testdurchführung beschrieben. Zum Schluss folgt die Auswertung des Eigenversuchs.
Die Beschreibung des von Studenten durchzuführenden Praktikumsversuchs zum Thema Testen ist Gegenstand des dritten Teils der Arbeit. Neben den dafür erforderlichen theoretischen Grundlagen, die in gestraffter Form nochmals dargestellt werden, stehen die Versuchsaufgaben eins bis drei im Mittelpunkt der Ausführungen.

2 Allgemeine Grundlagen

Bei einem Test soll durch einen Versuch oder eine Prüfung nachgewiesen werden, ob bestimmte Leistungsmerkmale des Testobjekts erfüllt werden. Es werden vorher festgelegte Sollwerte, die das Testobjekt erfüllen soll, mit den beim Test erhaltenen Istwerten verglichen. „Ein Test ist eine Aktivität, bei der ein Testobjekt mittels geeigneter Testmethoden statisch überprüft oder dynamisch ausgeführt wird, mit dem Ziel, Fehler aufzudecken und die korrekte und vollständige Umsetzung der funktionalen und nicht-funktionalen Anforderungen nachzuweisen.“[Franz, 2007] S. 24. Bezogen auf die Entwicklung von Softwaresystemen kann man Testen folgendermaßen beschreiben: „Testen ist ein experimentelles Verfahren, das mit einer beschränkten Zahl von Eingabekombinationen den empirischen Nachweis zu erbringen sucht, daß die Abweichung eines Programmes von seiner Spezifikation unterhalb einer vorgegebenen Toleranzschwelle liegt.“ [Balzert, 1992]. Beim Testen werden mehrere Ziele verfolgt [Spillner, Linz, 2005] S.9, und zwar:

- Ausführung des Programms mit dem Ziel, Fehlerwirkungen nachzuweisen.
- Ausführung des Programms mit dem Ziel, die Qualität zu bestimmen.
- Ausführung des Programms mit dem Ziel, Vertrauen in das Produkt zu erhöhen.
- Analysieren des Programms oder der Dokumente, um Fehlerwirkungen vorzubeugen.

Das Testen ist eine wichtige Phase in den Vorgehensmodellen zur Softwareentwicklung.

2.1 Vorgehensmodelle zur Softwareentwicklung

Software wird in ihrer Entwicklung und Wartung immer komplexer und aufwendiger. „Entwicklungsprozesse regeln die gesamten Arbeitsabläufe in der Software-Entwicklung. In einigen Anwendungsgebieten ist die Einhaltung bestimmter Entwicklungsprozesse verbindlich.“ [Liggesmeyer, 2009] S.362. Um die Entwicklung strukturiert und überschaubar zu machen, stehen mehrere Vorgehensmodelle zur Verfügung. In einem Vorgehensmodell wird der Entwicklungsprozess in zeitliche sowie inhaltliche Abschnitte unterteilt. Modelle sind unter anderem das Wasserfallmodell, das V-Modell, das Prototypenmodell und die Agile Softwareentwicklung. Folgende fünf Elemente finden sich in fast allen Modellen wieder [Rabe, Spickermann, Wenzel, 2008] S. 29:

- Aufgabenanalyse
- Modellformulierung

- Modellimplementierung
- Modellüberprüfung
- Modellanwendung

Zur geordneten Projektabwicklung definieren dabei alle Modelle eine Systematik. Es werden meistens Phasen- bzw. Arbeitsabschnitte festgelegt, die mit einem bestimmten Ergebnis in Form eines Dokuments abzuschließen sind. In jedem dieser Vorgehensmodelle findet sich das Testen wieder, allerdings mit sehr unterschiedlicher Bedeutung und mit unterschiedlichem Umfang.“ [Spillner, Linz, 2005].
Die verschiedenen Prozessmodelle werden nach folgendem Schema (Abbildung 1) eingeordnet.

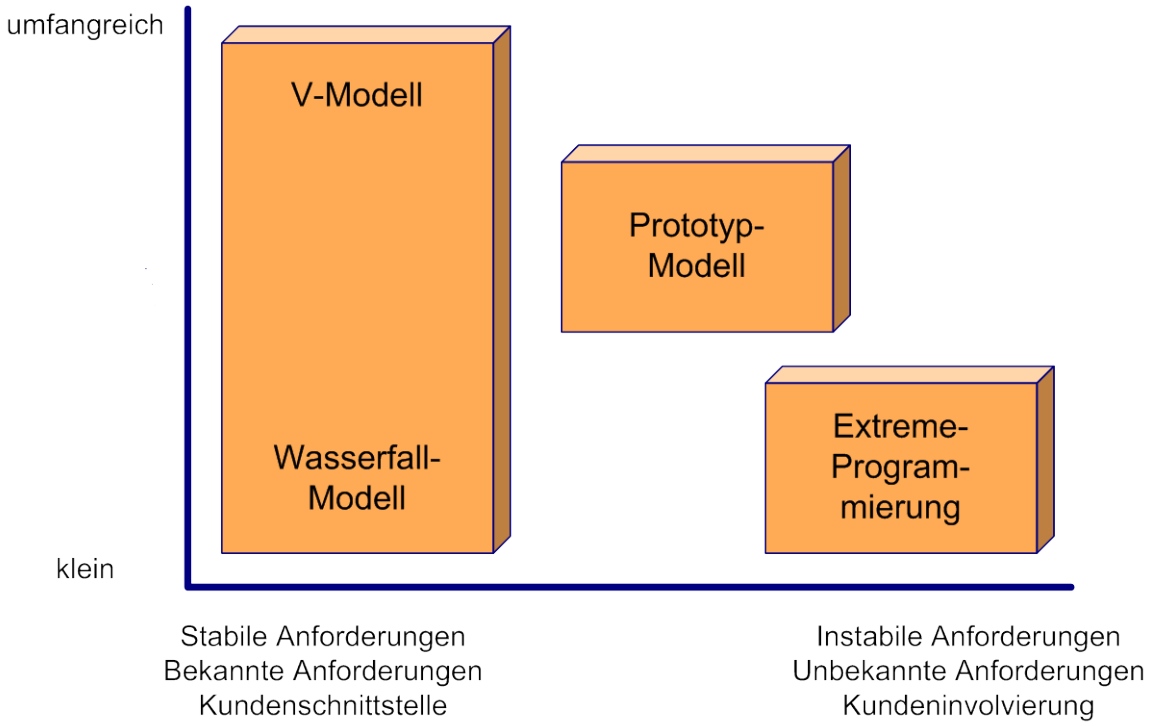

Abbildung 1: Einordnung von Prozessmodellen [Liggesmeyer, 2009]

Die Einordnung der Modelle erfolgt nach ihrer Eignung bezüglich des Projektumfangs, der Stabilität und Bekanntheit der Anforderungen sowie der Einbeziehung des Kunden. [Liggesmeyer, 2009]
Verschiedene Vorgehensmodelle zur Softwareentwicklung werden nachfolgend vorgestellt.

2.1.1 Wasserfallmodell

Als erstes grundlegendes Modell soll das Wasserfallmodell betrachtet werden. Viele umfangreiche Software-Produkte werden durch Entwicklungsprozesse erzeugt, denen ein Phasenmodell wie das Wasserfallmodell zugrunde liegen. „Der klassische Entwicklungsprozess folgt dem so genannten Wasserfallmodell. Dieses Modell sieht vor, dass der Entwicklungsprozess in Phasen aufgeteilt wird.“ [Liggesmeyer, 2009] S.363. Die einzelnen Phasen eines Wasserfallmodells für Protokollsoftware sind:

- Funktionsdefinition: Eine meist verbale Beschreibung der beabsichtigten Funktion.
- Dienstespezifikation: Eine Beschreibung, welche den Aufbau und das Verhalten des Systems bezogen auf die Dienste beschreibt.
- Protokollspezifikation: Eine Beschreibung, die den Aufbau und das Verhalten des Systems bezogen auf die Protokolle beschreibt.
- Implementierungsspezifikation: Ist eine implementierungsabhängige Beschreibung der Funktionalität eines Protokolls.
- Implementierung: Ist die Realisierung der beabsichtigten Funktionalität in einem System.
- Test: Ist die Überprüfung auf Vollständigkeit und Korrektheit einer Implementation.
- Betrieb/Wartung: Ist der Einsatz des Systems beim Kunden und die Betreuung des Systems vor Ort.

Der Abbildung 2 sind diese Phasen zu entnehmen.

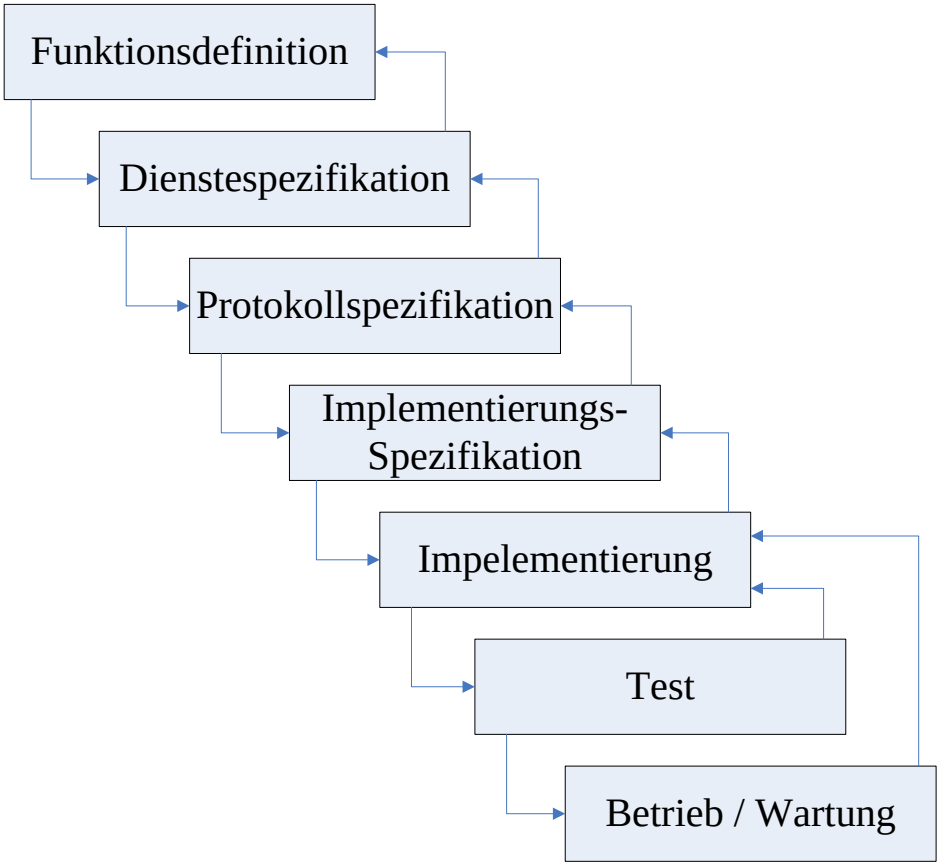

Abbildung 2: Wasserfallmodell

Eine Entwicklungsphase wird erst dann bearbeitet, wenn die Bearbeitung der vorherigen Phase abgeschlossen ist. Ein Rücksprung ist nur zwischen angrenzenden Phasen möglich. Ein umfassender Test wird erst vor der Inbetriebnahme des Systems durchgeführt. Auftretende Fehler werden dadurch spät erkannt. Die Beseitigung dieser Fehler ist dann zeit- und kostenaufwendig. Es wird daher oft als nachteilig empfunden, dass Modifikationen an dem zu entwickelnden Software-Produkt nur schwierig und mit hohem Aufwand möglich sind. Die Anforderungen müssen zu einem frühen Zeitpunkt festgelegt werden und können, nachdem sie festgeschrieben sind, kaum an geänderte Wünsche angepasst werden. Die Entwicklungsprozesse, die einem Phasenmodell folgen, haben für die Software-Qualitätssicherung den Vorteil, dass eine feste Prüfreferenz vorhanden ist, gegen die die Prüfung stattfinden kann. Darüber hinaus entstehen zu einem frühen Zeitpunkt Anforderungsdokumente – z.B. Lastenhefte –, welche als Grundlage eines Vertrages dienen können. „Aufgrund der expliziten Phasen erhält man eine entsprechende Abgrenzung der Tätigkeiten, die sich als Basis für eine arbeitsteilige Entwicklung anbietet. Die Software-Entwicklung wird in klar abgegrenzte Phasen unterteilt, die weiter in untergeordnete Arbeitspakete aufgeteilt werden. Diese nachvollziehbare Struktur begünstigt ein geeignetes Projektmanagement.“ [Liggesmeyer, 2009] S.363. Durch die deutliche Abgrenzung der einzelnen Entwicklungsphasen ist die Verfolgung des Projektverlaufes problemlos möglich.

2.1.2 V-Modell

Eine Erweiterung des Wasserfallmodells ist das V-Modell von Boehm. Das V-Modell (Abbildung 3) ergänzt das Wasserfallmodell um die Phasen Verifikation und Validierung. Die zwei Phasen sind von den konstruktiven Phasen getrennt und dienen einer ausführlicheren Püfung. Die Bezeichnung V-Modell stammt daher, dass das Modell grafisch in der Form des Buchstaben V dargestellt wird. Die linke Seite des V symbolisiert die Entwicklungsschritte, welche aus dem Wasserfallmodell bekannt sind. Die rechte Seite des V symbolisiert die Test- und Integrationsschritte. Zu diesen Schritten gehören die folgenden Tests:

- Der Unittest: Er überprüft, ob jedes einzelne Modul für sich die Vorgaben seiner Spezifikation erfüllt.
- Der Integrationstest: Er überprüft, ob die einzelnen Module, zu Gruppen zusammengefasst wie vorgesehen zusammenspielen.
- Der Systemtest: Er überprüft, ob das System als ganzes die geforderten Funktionen erfüllt.
- Der Abnahmetest: Er überprüft, ob das System die geforderten Funktionen aus Sicht des Kunden erfüllt.

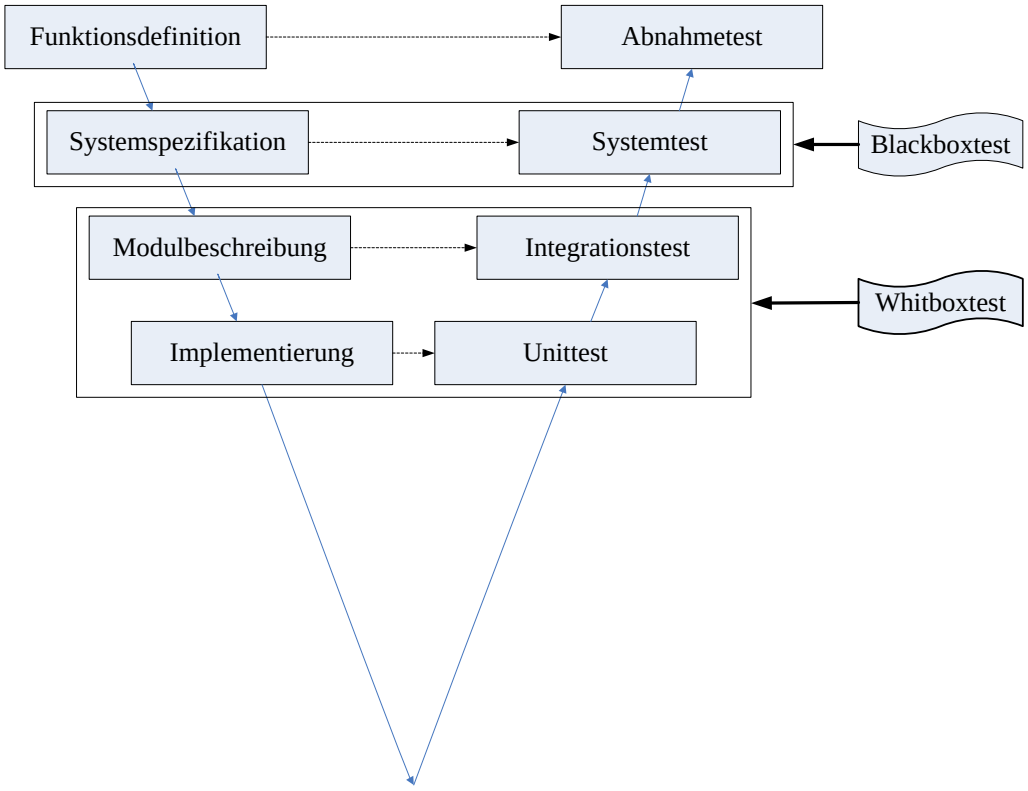

Abbildung 3: V-Modell

2.1.3 Prototypen-Modell

Eine Ergänzung zum Wasserfallmodell oder dem V-Modell ist das Prototypen-Modell. Prototypen sollen bei der Beseitigung von Unklarheiten z.B. im Hinblick auf Anforderungen oder die technische Machbarkeit helfen. Da in einigen Fällen eine abschließende Festlegung der gestellten Anforderungen in einer frühen Phase noch nicht möglich ist, ist die Anwendung einer prototypenorientierten Methode sinnvoll. Das Ziel des Prototypen-Modells ist, so schnell wie möglich funktionsfähige Modelle (Prototypen) zu erstellen, mit denen die Ideen geprüft werden können. Der Anwender soll so früh wie möglich mit den lauffähigen Modellen konfrontiert werden, um den zeitlichen Aufwand, Fehler zu beseitigen bzw. Änderungen einzubringen, gering zu halten.
Bei der Anwendung des Prototypen-Modells unterscheidet man zwischen „Wegwerfprototypen“ und „evolutionären Prototypen“. Die „Wegwerfprototypen werden mit speziellen Entwicklungsmethoden erstellt. Diese „Wegwerfprototypen“ entsprechen nicht dem eigentlichen angestrebten System. Sie sollen aber eine schnelle Realisierung gewährleisten, die an das Zielsystem heranführen. Ist das Ziel erreicht, so wird der Prototyp „weggeworfen“ und die Entwicklung beginnt von neuem. Die evolutionären Prototypen werden bis zum vollständigen Produkt entwickelt. Der Prototyp wird dabei nicht verworfen. [Liggesmeyer, 2009]

2.1.4 Agile Softwareentwicklung

Die Agile Softwareentwicklung hebt sich vom Wasserfallmodell und dessen Weiterentwicklungen ab. Der Begriff Agil kommt aus dem lateinischen und bedeutet flink bzw. schnell. Das Ziel dieser Methode ist, die Entwicklungsprozesse flexibler zu gestalten. Kennzeichnend ist, dass sich Agile Methoden nicht den schwergewichtigen Prozessen und deren festgelegten Vorgehensweisen zuordnen lassen. Der bürokratische Aufwand soll stark minimiert werden. Anstelle des Planens wird die lauffähige Software in den Vordergrund gestellt. Ein wichtiger Bestandteil dieser Entwicklungsmethode ist die enge Zusammenarbeit mit dem Kunden. Für den Anwender besteht die Möglichkeit, auch während der Umsetzung Einfluss auf die Produktgestaltung zu nehmen. Durch geringeren bürokratischen Aufwand wird der Entwicklungsprozess flexibler. Im Jahr 2001 wurde das Agile Manifesto veröffentlicht, mit dem Ziel Entwicklungsprozesse flexibler zu gestalten. 17 Personen, welche auf dem Gebiet der Softwareentwicklung tätig sind, unterzeichneten das Manifest mit den folgenden Aussagen [infforum 2009]:

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionierende Software ist wichtiger als umfassende Dokumentation.
- Kundenzusammenarbeit ist wichtiger als Vertragsverhandlungen.

- Auf Änderungen reagieren ist wichtiger, als einem Plan zu folgen.

Die durch das Agile Manifesto festgelegten Methoden für Softwareentwicklung, welche die unterzeichnenden Personen anwenden, sind in zwölf Prinzipien beschrieben. [infforum 2009]

1. Stelle den Kunden durch frühzeitige und regelmäßige Auslieferung nützlicher Software zufrieden.
2. Sich ändernde Anforderungen werden begrüßt, selbst wenn diese spät in der Entwicklung auftreten. Agile Prozesse machen Änderungen als Wettbewerbsvorteil für Kunden nutzbar.
3. Lauffähige Software soll häufig, in Abständen von wenigen Wochen bis Monaten ausgeliefert werden. Kurze Zeitspannen sollten bevorzugt werden.
4. Geschäftsleute und Entwickler müssen während des gesamten Projektes täglich zusammenarbeiten.
5. Baue Projekte um motivierte Individuen herum auf. Stell ihnen die benötigte Umgebung und Unterstützung zur Verfügung und vertraue auf sie.
6. Die effizienteste und effektivste Methode, um Wissen innerhalb des Entwicklerteams zu verteilen oder an das Team heranzutragen, ist ein persönliches Gespräch von Angesicht zu Angesicht.
7. Lauffähige Software ist das wichtigste Maß für den Projektfortschritt.
8. Agile Prozesse fördern die dauerhafte Entwicklung. Geldgeber Entwickler und Nutzer sollten in der Lage sein, eine konstante Fortschrittsgeschwindigkeit dauerhaft beizubehalten.
9. Ständige Aufmerksamkeit auf die technische Güte und gutes Design verbessert die Agilität.
10. Einfachheit – die Kunst die Menge der Arbeit, die nicht getan wird, zu maximieren – ist entscheidend.
11. Selbstorganisierende Teams sind der Schlüssel zu guten Architekturen, Anforderungsspezifikationen und Entwürfen.
12. In regelmäßigen Abständen sollte das Team überlegen, wie es effektiver werden kann und dann sein Verhalten entsprechend ändern und anpassen.

Für die Autoren, welche das Manifest unterzeichneten, ist die Kommunikation im Team und die Kommunikation mit dem Kunden die notwendige Voraussetzung für den Erfolg eines Projektes. Der Erfolg wird daran gemessen, ob Software erstellt wird, die die sich ändernden Anforderungen des Kunden erfüllt.

2.1.5 Extreme Programmierung (XP)

Die extreme Programmierung ist eine der bekanntesten agilen Softwareentwicklungsmethoden. Anwendung findet die extreme Programmierung bei kleineren Projekten, bei denen die Anforderungen noch nicht vollständig festgelegt sind. Der Kunde muss bereit sein, aktiv mitzuarbeiten. Im Vordergrund steht die starke Einbeziehung des Kunden. Durch die extreme Programmierung wird eine Entwicklung in schnellen Iterationen erreicht. Als Iteration wird in der Softwaretechnik ein einzelner Entwicklungszyklus bezeichnet. Neue Funktionalitäten werden ständig entwickelt, integriert und getestet. Qualitätssichernde Maßnahmen während des gesamten Projektes, sollen hinreichende Qualität gewährleisten.
Bei der Anwendung der extremen Programmierung ist das Ziel, die Software in sehr hoher Qualität und schneller an den Kunden auszuliefern. Der Kunde erhält ein lauffähiges Produkt, an dessen Herstellung er aktiv teilgenommen hat.

2.1.6 Fazit

Jedes Vorgehensmodell hat seine Vor- und Nachteile. Die Modelle haben sich im Laufe der Zeit weiterentwickelt. Es muss jeweils entschieden werden, welche Merkmale Priorität für die Aufgabenstellung haben. Danach sollte das Modell gewählt werden.

2.2 Testfallentwurfsverfahren

Testtechniken können nach unterschiedlichen Kriterien klassifiziert werden. Möglichkeiten der Klassifizierung sind die Einteilung in Whiteboxtest und Blackboxtest oder in funktionale und nichtfunktionale Tests.

2.2.1 Whiteboxtest und Blackboxtest

Die Unterscheidung in Whiteboxtest und Blackboxtest ist weit verbreitet. Die Einteilung muss als zu grob nach dem heutigen Stand der Technik betrachtet werden. [Liggesmeyer 2009]. Testtechniken, die sehr verschieden sind, werden den Whiteboxtests bzw. Blackoxtests zugeordnet.

2.2.1.1 Whiteboxtest

Der Whiteboxtest ist neben dem Blackboxtest eine Methode zum Testen von Software. Whiteboxtests orientieren sich dabei am Quelltext der zu testenden Software. Grundlage der Whiteboxtests ist der Quelltext des Testobjekts. Die Testverfahren werden daher oft als

codebasierte Testverfahren bezeichnet. Der Quelltext muss vorhanden sein und unter Umständen auch manipuliert, d. h. ergänzt werden können. Die grundlegende Idee der Whiteboxtests ist, dass alle Quellcodeteile eines Testobjekts mindestens einmal ausgeführt werden sollen. „Aufgrund der Programmlogik werden ablauforientierte Testfälle ermittelt und ausgeführt. Dabei ist selbstverständlich auch die Spezifikation zu berücksichtigen, um den Testfall zu erstellen und insbesondere nach seiner Ausführung bewerten und entscheiden zu können, ob ein fehlerhaftes Verhalten vorliegt.“ [Spillner, Linz, 2005] S. 144 Whiteboxtests allein sind als nicht ausreichend anzusehen. Sie sind gut geeignet, um beispielsweise Fehler in einzelnen Komponenten aufzuspüren. Zu den Whiteboxtests gehören unter anderem strukturorientierte Testtechniken.

2.2.1.2 Blackboxtest

Eine weitere Methode, welche betrachtet werden soll, ist der Blackboxtest. Dabei wird überprüft, ob ein System mit seiner Spezifikation übereinstimmt. Bei den Blackboxtests wird das Testobjekt als schwarzer Kasten angesehen. Informationen über den Quelltext und den inneren Aufbau sind nicht notwendig. Beobachtet wird das Testobjekt von außen (PoO-Point of Observation liegt außerhalb des Testobjekts). Eine Steuerung des Ablaufs des Tests ist nur von außer durch die entsprechende Wahl der Eingabetestdaten möglich (auch der PoC-Point of Control liegt außerhalb des Testobjekts). Durch die Spezifikation bzw. die Anforderungen an das Testobjekt, werden die Testfälle erstellt. „Blackbox-Verfahren eignen sich überwiegend für die höheren Teststufen, wobei auch beim Komponententest ein Einsatz sinnvoll ist. Alle Vorgehensmodelle, bei denen vor der Kodierung die Testfälle definiert werden *(test-first programming, test-driven development)* sind Blackbox-Verfahren.“ [Spillner, Linz, 2005] S. 108-109.
Zu den Blackboxtests gehören unter anderem die Protokolltests, die Funktional-Tests, die Kompatibilitätstests und die Performancetests.

2.2.1.3 Fazit zu Whiteboxtsts und Blackboxtests

Die Vorteile von Whiteboxtests und Blackboxtests werden in den Grayboxtests zusammengefasst. Es sollen die Nachteile vermieden werden, die Blackboxtests und Whiteboxtests haben. Beispielsweise ist ein Nachteil des Blackboxtest, dass das Arbeiten mit dem Quellcode nicht möglich ist. Bei einem Grayboxtest besteht die Möglichkeit, am Quellcode zu arbeiten.
Wie schon eingehend erwähnt ist die Einteilung von Tests in Whiteboxtests und Blackboxtests unzureichend. Es gibt bessere Möglichkeiten Tests einzuordnen, z.B. in funktionale und nichtfunktionale Tests.

2.2.2 Funktionale und nichtfunktionale Tests

Eine weitere Möglichkeit der Klassifizierung ist die Einteilung in funktionale und nichtfunktionale Tests.

2.2.2.1 Funktionale Tests

Die funktionalen Tests betrachten die korrekte Funktion eines Systems. Testfälle werden anhand der Spezifikation gebildet. Geprüft wird die Übereinstimmung des Systems mit der Spezifikation. Funktionale Tests werden nach zwei Kriterien unterschieden. Das erste Kriterium richtet sich nach der verfügbaren bzw. benutzten Information des zu testenden Systems. Das zweite Kriterium richtet sich nach unterschiedlichen Strategien, wie viele ausreichende und repräsentative Testfälle gefunden werden. Das gemeinsame Ziel ist, das Verhalten eines Systems zu prüfen. Dies soll in einem vertretbaren Aufwand geschehen, da es unverhältnismäßig ist alle möglichen Fälle, die auftreten könnten, zu untersuchen. Die Auswahl an Testfällen soll möglichst repräsentativ für die Menge aller möglichen Eingaben sein. Grundlage sind die Schnittstellen des zu testenden Systems mit dessen Ein- und Ausgabeparametern. Testfälle sollten so definiert werden, dass [qse, 2009]:

- sie mit einer hohen Wahrscheinlichkeit Fehler finden, d.h. dass sie einen großen Teil des Systems überdecken und eine hohe Anzahl von wichtigen und verschiedenen Fehlern finden,
- so wenige Aktionen wie möglich doppelt ausgeführt werden müssen, d.h. dass sie so billig wie möglich sind,
- sie so detailliert wie nötig sind, aber so viele Freiheiten wie möglich bieten,
- der Prozess der Definition selbst so billig wie möglich ist.

2.2.2.2 Nichtfunktionale Tests

Die Zuverlässigkeit eines Systems hängt nicht allein von der korrekten Funktion ab. Nichtfunktionale Tests untersuchen Systeme in Bezug auf Performance, der Benutzbarkeit und der Sicherheit.
Ein Beispiel für nichtfunktionale Tests ist der Performancetest, welcher unter Punkt 2.2.3.6 ausführlicher beschrieben wird.
Auch die Sicherheit in den Systemen z.B. gegen unerlaubten Zugriff von außen ist ein wichtiger Teil, der durch nichtfunktionale Tests geprüft wird. Durch einen Penetrationstest kann die Überprüfung der Sicherheit eines Systems durchgeführt werden. Durch das Testsystem

wird das zu überprüfende System angegriffen. Die Angriffe sind aus vergleichbaren Systemen, die als Referenzsystem dienen, bekannt.
Nichtfunktionale Tests, welche die Benutzbarkeit untersuchen sind aufwendig. Oft werden diese Tests durch Benutzer an Prototypen durchgeführt.

2.2.3 Testtechniken

Es existieren verschiedene Testtechniken mit zum Teil unterschiedlichen Zielstellungen bezogen darauf, was getestet werden soll. Einige Testtechniken werden nachfolgend vorgestellt.

2.2.3.1 Überdeckungstests (Codecoveragetests)

Überdeckungstests sind kontrollflussorientierte Testtechniken, die den Whiteboxtests zugeordnet werden, da mit dem Quellcode gearbeitet wird. Es wird von überdecken gesprochen, da die Prorammteile, die getestet werden, durch den Test abgedeckt werden. Bei den Überdeckungstests wird der Quellcode zur Ausführung gebracht. Die wichtigsten Überdeckungstests sind:

- Anweisungsüberdeckung: Die Anweisungsüberdeckung ist eine der einfachsten Überdeckungstests. Durch die Anweisungsüberdeckung sollen Teile im Programm ausfindig gemacht werden, die niemals durchlaufen werden. Jede Anweisung soll durch den Test wenigstens einmal zur Ausführung gebracht werden. [Liggesmeyer 2009].
- Zweigüberdeckung: Bei der Zweigüberdeckung werden die Kanten und Zweige in einem Kontrollflussgraphen untersucht. Alle Zweige werden durchlaufen. Dadurch können nichtausführbare Zweige aufgespürt und untersucht werden. Bei einer Verzweigung im Kontrollfluss müssen alle Möglichkeiten und bei Schleifen neben dem Schleifenkörper auch die Umgebung bzw. der Rücksprung zum Schleifenanfang untersucht werden. [Spillner, Linz, 2005].
- Zeilenüberdeckung: Im Zeilenüberdeckungstest sollen alle Zeilen des Quellcodes zur Ausführung gebracht werden. Die Anweisungen werden nicht betrachtet. [Wikipedia 2009].
- Pfadüberdeckung: Durch die Pfadüberdeckung sollen alle möglichen Pfade der zu testenden Software ausgeführt werden. Betrachtet werden alle Pfade vom Start bis zum Ende. [Liggesmeyer 2009].

- Bedingungsüberdeckung: Die Bedingungsüberdeckung betrachtet die logische Struktur von Entscheidungen der zu testenden Software.[Liggesmeyer 2009].

Überdeckungstests liefern Werte der strukturellen Testabdeckung während der Testdurchführung. „Hierzu werden vor der Testdurchführung (von einer Instrumentierungskomponente des Analysewerkzeugs) Messanweisungen in das Testobjekt eingeführt (Instrumentierung). Wird im Testlauf eine solche Messanweisung ausgelöst, wird die entsprechende Programmstelle als »überdeckt« protokolliert.“ [Spillner, Linz, 2005] S. 213.
Das qualitative Ergebnis eines Überdeckungstests hängt davon ab, welcher Test gewählt wird. Beispielsweise kann ein Anweisungsüberdeckungstest, der vollständig ausgeführt wurde, nicht garantieren, dass das Programm fehlerfrei ist. Ein hohes qualitatives Ergebnis eines Überdeckungstests erfordert einen hohen Zeitaufwand und damit verbundene Kosten.

2.2.3.2 Unittest Integrationstest

Der Unittest und der Integrationstest sind Beispiele für Whiteboxtests. Sie werden auf den unteren Teststufen angewendet. Nachfolgend sollen der Unittest (auch Komponententest, Modultest oder Klassentest) und der Integrationstest etwas näher erläutert werden. Ein Merkmal für den Komponententest ist, dass jeweils ein einzelner Softwarebaustein überprüft wird, und zwar separat von den anderen Softwarebausteinen des Systems. Diese Abgrenzung hat dabei den Zweck, komponentenexterne Einflüsse beim Test zu vermeiden. Deckt der Test einen Fehler auf, lässt sich dessen Ursache dann klar der getesteten Komponente zuordnen. Das entscheidende ist, dass komponenteninterne Aspekte geprüft werden, jedoch nicht die Wechselwirkung mit den Nachbarkomponenten. Das ist Bestandteil des Integrationstests. [Spillner, Linz, 2005]. Die einzelnen Komponenten werden gegen ihre Spezifikation geprüft. Diese Spezifikationen sind das Ergebnis des Feinentwurfs. Alle Feinheiten der einzelnen Komponente sind von Bedeutung. [Liggesmeyer, 2009].
Nach erfolgreich abgeschlossenem Unittest folgt der Integrationstest. Bei dem werden einzelne Units (auch Komponenten oder Module) zusammengefasst und als gesamte Gruppe getestet. Durch den Integrationstest soll das Interagieren der verschiedenen Komponenten über die Schnittstellen überprüft werden. Gruppen dieser Komponenten werden dann von Entwicklern, Testern oder speziellen Integrationsteams zu größeren Baugruppen bzw. Teilsystemen zusammengefasst. Dies wird als Integration bezeichnet. Im Anschluss daran muss getestet werden, ob das Zusammenspiel aller Einzelteile miteinander korrekt funktioniert. Der Integrationstest hat demzufolge das Ziel, Fehlerzustände in Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten zu finden. [Spillner, Linz, 2005]. In der Praxis

wird der Integrationstest oft in mehrere untergeordnete Phasen aufgeteilt. Dadurch wird eine sinnvolle Prüfung auch großer Systeme ermöglicht. [Liggesmeyer, 2009].
Die Überprüfung einzelner Komponenten und die anschließende Integration ist eine empfehlenswerte Vorgehensweise. Auftretende Fehler werden frühzeitig in den Komponenten aufgespürt. Die Suche eines Fehlers in zusammengefassten Baugruppen, der sich auf eine Einzelkomponente bezieht, kann vermieden werden.

2.2.3.3 Protokolltest

Durch den Protokolltest werden die verwendeten Protokolle geprüft, ob sie die Vorgaben der Spezifikation erfüllen. Protokolle sind Vereinbarungen, die die Kommunikation zwischen den miteinander arbeitenden Systemen regeln.
Zu den Protokolltests gehört der Konformitätstest, mit dem festgestellt wird, ob eine Implementierung mit den Spezifikationen des Standards übereinstimmt. Mit dem Konformitätstest überprüft man eine Implementation in Bezug auf die im entsprechenden internationalen Standard oder bezüglich der in den ITU-T-Empfehlungen beschriebenen Konformitätsanforderungen. [ITWissen, 2009]. Das Ziel des Tests ist zu prüfen, ob die verschiedenen Implementierungen der Systeme zuverlässig zusammenarbeiten. Auch bei diesem Test ist aufgrund der Komplexität der Protokolle eine ausführliche Testung unwirtschaftlich und damit kaum durchführbar. Der Standard ISO IS 9646 beschreibt die Vorgehensweise für den Konformitätstest und die Notationsform TTCN zur Erstellung von Protokolltests.

2.2.3.4 Funktionaltest

Bei den Funktional-Tests oder auch Funktionstests wird die Funktionalität der Kommunikationssoftware getestet. Beim Funktionstest ist eine funktionsorientierte Testplanung durchzuführen. Die Anforderungsdokumente, die systematisch in Testfälle umzusetzen sind, bilden die Basis für diese Testplanung. [Liggesmeyer, 2009]. Es wird die Übereinstimmung der Funktionalität mit einem vorher beschriebenen Szenario überprüft. Dieses Szenario kann vom Kunden beschrieben werden. Es ist ein Test gegen eine bestimmte Spezifikation. Ein Beispiel für Funktional-Tests ist die Äquivalenzklassenbildung.

2.2.3.5 Kompatibilitätstest

Der Kompatibilitätstest soll feststellen, ob Hard- und Softwarekomponenten verschiedener Hersteller auf Schnittstellenebene problemlos miteinander zusammenspielen. Dazu gehört auch die Verträglichkeitsprüfung mit vorhandenen Systemen. Ein komplexes Kommunikationssystem muss nicht aus Komponenten nur eines Herstellers bestehen. Es besteht da-

durch die Anforderung, dass die einzelnen Komponenten miteinander ohne Einschränkungen funktionieren. Zur Kompatibilität tragen Standards bei. Diese sichern, dass z.B. internationale Schnittstellen, die Komponenten miteinander verbinden, gleich gestaltet sind. Kompatibilität kann auf- oder abwärtsgerichtet sein. Bei der Abwärtskompatibilität können neue Systeme mit älteren Systemen zusammenarbeiten. Aufwärtskompatibilität bedeutet, dass ein System auch mit zukünftig entwickelten Systemen kompatibel sein wird.

2.2.3.6 Performancetest (Leistungstest)

Der Performancetest bzw. Leistungstest ist eine Untersuchung, bei der das Verhalten des Systems unter Belastung geprüft wird. Es wird simuliert, dass tausende Zugriffe auf das Kommunikationssystem gestartet werden. Dabei werden Tests durchgeführt, die das Verhalten des Systems oberhalb der Anforderungen untersuchen. Damit sollen Erkenntnisse darüber gewonnen werden, wie das System auf außergewöhnliche Situationen reagiert. „Der Leistungstest bringt das Software-System an den Grenzbereich heran, jedoch nicht darüber hinaus. Zu diesem Zweck müssen Lasten erzeugt, und gleichzeitig z.B. Zeiten und Auslastung gemessen werden können. Dies ist manuell in der Regel nicht möglich. Die Durchführung von Leistungstest ist daher auf eine geeignete Werkzeugunterstützung angewiesen.“ [Liggesmeyer, 2009] S. 377.

2.3 Testwerkzeuge

Testwerkzeuge sollen die Software-Prüfung unterstützen. Sie können eine geeignete Testmethodik nicht ersetzen, jedoch können sie sie unterstützen. Die Auswahl geeigneter Testmethoden und Techniken steht deshalb an erster Stelle. Die Auswahl von Werkzeugen muss die ausgewählten Methoden und Techniken berücksichtigen. Der Kauf eines Werkzeuges, ohne zuvor die geeignete methodische und technische Vorgehensweise festgelegt zu haben, ist unsinnig. Dennoch sind Werkzeuge für die Unterstützung des Tests wichtig und für manche Techniken unverzichtbar. Werkzeuge sollen bei der effizienten Durchführung von Prüfungen ein nützliches Hilfsmittel sein. Sie geben Auskunft über den Stand der Prüfung. Sie nehmen zeitaufwendig Auswertungen vor. Darüber hinaus soll die werkzeugunterstützte Protokollierung einer Prüfung unverzichtbarer Bestandteil für den Prüfungsnachweis gegenüber Kunden sein. [Liggesmeyer, 2009].
Nachfolgend sollen beispielhaft Testwerkzeuge für Whiteboxtests und Blackboxtests kurz vorgestellt werden.

2.3.1 Whiteboxtestwerkzeuge

2.3.1.1 Debugger

Durch einen Debugger sollen Programmierfehler gefunden werden. Der Debugger lässt das Programm kontrolliert ablaufen und untersucht dieses dabei auf Funktion und Logik. Schon während der Entwicklung eines Systems wird eine Software eingesetzt, die die Bugs also Fehler aufspüren soll. Durch einen Debugger kann man den Inhalt von verwendeten Variablen in einem Programm untersuchen. Weiterhin besteht die Möglichkeit, durch die schrittweise Ausführung des Systems, Fehler im Ablauf aufzuspüren. Ein Vertreter der Debugger ist der GNU-Debugger. Er ist ein Standard-Debugger für Linux-Systeme.

2.3.1.2 Profiler

Ein Profiler ist ein Werkzeug, mit dem Laufzeituntersuchungen an Software analysiert werden können. Dabei wird ermittelt wie oft Programmteile ausgeführt werden. Weiterhin wird gemessen, wie viel Zeit für die Ausführung benötigt wird. Dadurch können im Programm die Stellen ausfindig gemacht werden, die den Programmablauf ausbremsen. An diesen Stellen kann der Softwareentwickler ansetzen, den Ablauf zu optimieren.

2.3.1.3 Coverage

Ein weiteres Beispiel für ein Whiteboxtestwerkzeug ist der Coveragetest. Unter Punkt 2.2.3.1 S.13 wurde die Funktionsweise der Coveragetests beschrieben. Ein Beispiel für einen Coveragetest ist Testwell CTC++ (Test Coverage Analyser für C und C++). Mit diesem Werkzeug können Coveragetests in den Programmiersprachen C, C++, Java und C# durchgeführt werden. Mit Testwell CTC++ werden alle Testabdeckungsstufen analysiert. Dadurch ist er für Tests mit hohen Anforderungen geeignet. [Verifysoft, 2010].

2.3.1.4 Unitframework

Als Unitfamwork werden automatische Softwaretests bezeichnet. Durch die Frameworks können verschiedene Units hinsichtlich ihrer Funktion überprüft werden.
JUnit ist ein Frameworktestwerkzeug, welches zum systematischen testen von Java-Programmen dient. JUnit ist besonders für automatisierte Unittests geeignet und kennt nur zwei Ergebnisse. Zum einen, dass der Test erfolgreich ist und zum anderen, dass der Test misslingt. [Wikipedia, 2009].

2.3.2 Blackboxtestwerkzeuge

Auf dem Markt werden eine Vielzahl von Testwerkzeugen für Blackboxtests angeboten. Zwei Testwerkzeuge für Blackboxtests sind TTCN und IPS.
Zur Lösung der gestellten Aufgaben, findet das IPS-Werkzeug Anwendung.

2.3.2.1 TTCN

TTCN (Tree and Tabular Combined Notation) ist eine Sprache, mit der Protokolle im Bereich der Kommunikationstechnik getestet werden können. Diese Sprache ist standardisiert. Nachfolgeversionen von TTCN sind TTCN-2 und TTCN-3. Die Abkürzung TTCN-3 steht für Testing and Test Control Notation. Diese Version entspricht dem aktuellen Stand.
TTCN-3 ist gegenüber TTCN universeller und wir dadurch in vielen Bereichen eingesetzt. Neue Bereiche, in denen die Sprache Anwendung findet sind die Automobilindustrie, die Eisenbahntechnik, die Luftfahrttechnik und die Sicherheitstechnik. Durch die Sprache wird eine Vielzahl von Blackboxtests unterstützt. Herausgeber von TTCN-3 ist das europäische Institut für Telekommunikationsnormen (ETSI). TTCN-3 ähnelt einer typischen Programmiersprache. Zusätzlich besitzt sie wichtige Sprachmittel, die für die Spezifikation von Testfällen für Funktions- oder Performancetests notwendig sind. [software-kompetenz, 2009]. Die Sprache wird immer weiterentwickelt und verbessert. Dadurch wird sie an neue Anforderungen angepasst.

2.3.2.2 IPS

IPS (Independent Protocol Simulator) ist ein Programm zur Simulation von Protokollen. Durch das Programm können Abläufe analysiert und Systeme untersucht werden. IPS wurde durch die Firma Nokia-Siemens-Networks entwickelt. Es bietet eine Vielzahl von Testmöglichkeiten wie z.B. Funktions- und Performancetests. IPS ist ein Programm mit einer grafischen Benutzeroberfläche, welches unter Windows läuft. Diese Benutzeroberfläche gibt Auskunft über den Fortschritt der Simulation, die tatsächliche Belastung und die Fehlerquote. Bestandteil dieser Benutzeroberfläche ist der Trace-Teil im IPS-Fenster (Abbildung 4). Dort werden alle wichtigen Informationen zur Bewertung der Simulation automatisch generiert. Der Umfang kann durch den Benutzer festgelegt werden.

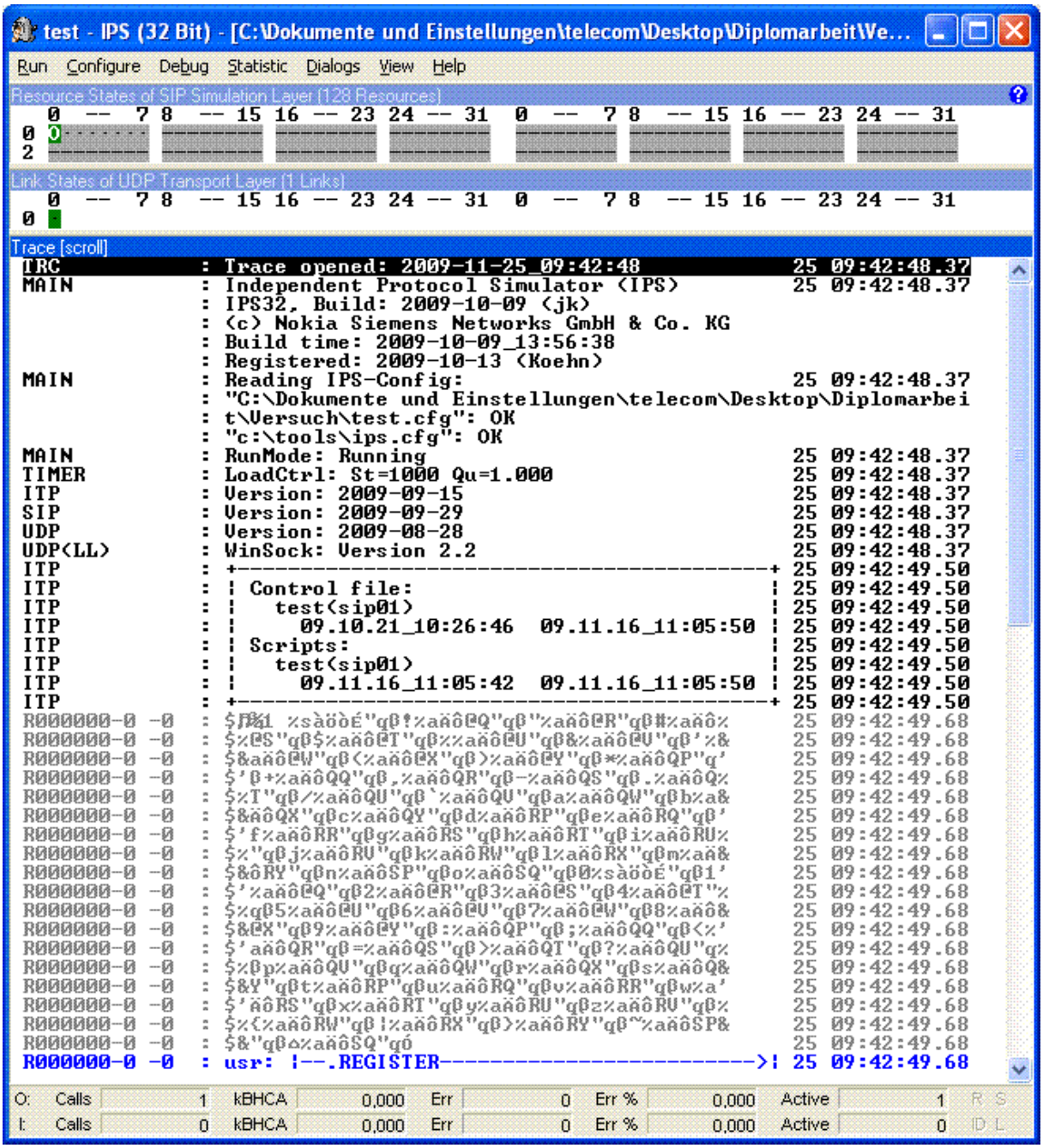

Abbildung 4: IPS-Fenster

Weiterhin besteht die Möglichkeit, interaktiv Einfluss auf den Testverlauf zu nehmen (stop load, control load, start test on key). IPS unterstützt die parallele Ausführung von Testfällen. Das ist besonders bei Perormancetests oder Stresstests von Bedeutung. Es besteht zum Beispiel die Möglichkeit, mehrere Verbindungen gleichzeitig zur Ausführung zu bringen. Die Ausführung erfolgt mehrmals parallel oder in Kombination beider. Das IPS-Programm ist modular aufgebaut. Das bedeutet, dass einzelne Funktionen in einem Modul untergebracht sind. In der Abbildung 5 wird diese modulare Struktur deutlich.

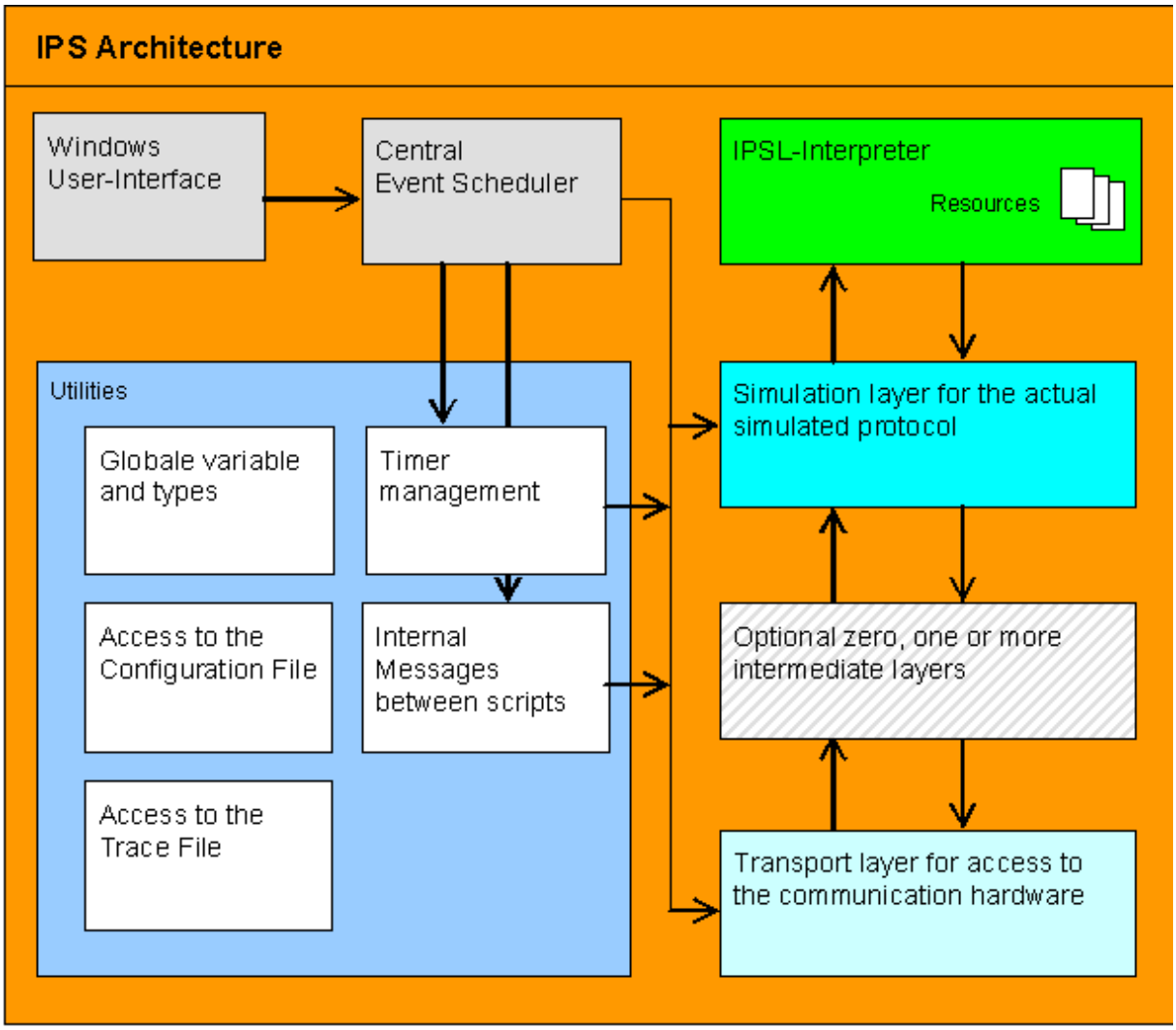

Abbildung 5: IPS Architektur [Nokia-Siemens-Networks, 2009]

Das Utilitie-Modul enthält die Dienstprogramme mit allgemeinen Aufgaben z.B. das Timermanagement. Weitere Module sind Simulation layer (Simulationsschicht), Intermediate layer (Zwischenschicht) und Transport layer (Transportschicht).
Die Simulationsschicht hat unter anderem die Aufgaben, die Nachrichten aus den Daten einer unteren Schicht zu selektieren, den Typ der Nachricht zu ermitteln und die Nachricht an den Traceteil zu schicken. Die Nachrichten werden entweder in PDUs verpackt oder aus PDUs gewonnen. An die Simulationsschicht schließt sich die Zwischenschicht an. Diese Schicht ist optional. Es kann keine, eine oder mehrere Zwischenschichten geben. Die Zwischenschicht ist verantwortlich für die Formatierung der Nachrichten und die Bereitstellung spezieller optionaler Services. Die unterste Schicht ist die Transportschicht. Diese Schicht stellt die Anbindung an das Betriebssystem bereit und bietet eine Reihe von Kommunikationskanälen (Links) zu den höheren Schichten. Weiterhin erfolgt über sie die Berichterstattung an die höheren Schichten. [Nokia-Siemens-Networks, 2009].

Die Module sind in verschiedenen Varianten in IPS enthalten. Welches von ihnen benutzt wird, ist in der Konfigurationsdatei (config file) festgelegt.

2.4 Sprachübertragung durch VoIP

-VoIP

VoIP steht für Voice over IP also Sprachübertragung über Computernetzwerke, z.B. das Internet. Die Sprachinformationen werden über ein Netz geleitet, das ansonsten für die Datenübertragung genutzt wird. Sozusagen ein Netz für alle Dienste. Der Vorteil von VoIP besteht darin, dass verschiedene Dienste über ein und dieselbe Leitung genutzt werden können. Solche Dienste können beispielsweise Telefonie und Datenübertragung sein. Nachteilig bei VoIP ist, dass sich Codierungs- bzw. Decodierungsverzögerungen ergeben können und dass das Thema Notruf noch nicht geklärt ist. Das Problem dabei ist, dass beim Absetzen eines Notrufes die Positionsdaten nicht mitgeteilt werden.
Im Gegensatz zur Telefonie über das Fernsprechnetz wird die Nachricht bei VoIP paketvermittelt. Es gibt Standards, durch welche VoIP geregelt wird. Die Standardisierung erfolgt hauptsächlich durch die ITU-T, die IETF und das ETSI. Die drei Organisationen haben unabhängig voneinander eigene Standards entwickelt. Durch die ITU werden grundlegende Verfahren und Komponenten standardisiert, und zwar in der:

- H-Serie (Audiovisual and multimedia systems): Einsatz von Leitungen für Nicht-Telefonie.
- G-Serie (Transmission systems and media, digital systems and networks): Fernsprechübertragung über Draht, Funk, Satelliten und digitale Netze.
- Q-Serie (Switching and signalling): Fernsprechzeichengabe und Vermittlung.

Die IETF regelt RTP, SIP und SDP. Durch das ETSI wird die Zusammenarbeit zwischen herkömmlichen Netzen und Internetanwendungen geregelt. [Vorlesung, 2009]
Die Möglichkeiten der Teilnehmeradressierung sind vielfältig. Eine Adresse besteht aus einem User-Teil und einem Host-Teil, z.B. *sip:meier@htwm.de*, *sip:+49-3727-581404@htwm.de* oder auch *sip:meier@141.55.xxx.xxx*.
Eine Kommunikationsbeziehung wird in drei Phasen unterteilt:

- Sitzungsaufbau
- Nutzung
- Sitzungsabbau

Der Sitzungsauf und –abbau wird durch SIP geregelt. Die Sprachsignale der Nutzung werden mittels Real Time Protocol (RTP) verschickt. Das Real Time Protocol sorgt dafür, dass die Multimediadaten in Echtzeit übertragen werden

-SIP

Die Abkürzung SIP steht für Session Initiation Protocol. Dabei handelt es sich um ein Signalisierungsprotokoll. Das Protokoll wird für den Aufbau, den Abbau und zur Steuerung einer Multimediasitzung zwischen zwei Teilnehmern verwendet. Das von der IETF entwickelte Session Initiation Protocol wird im RFC 3261 definiert. [IETF, 2009]. SIP ist ein offenes Protokoll und hat im Bereich Voice over IP seinen festen Platz gefunden. Das Session Initiation Protocol ist in seiner Struktur ähnlich dem Hypertext Transfer Protocol. Die Kommunikation bei SIP erfolgt über Nachrichten (Messages). Diese Nachrichten werden zwischen den Clients und Servern ausgetauscht. Es werden zwei Arten von Nachrichten unterschieden: Requests (Anfrage, Anforderung) und Responses (Antworten). Die Requests werden vom Clint erzeugt und an den Server geschickt. Responses werden vom Server erzeugt und an den Clint übermittelt. Die Abbildung 6 verdeutlicht den Aufbau der Nachrichten.

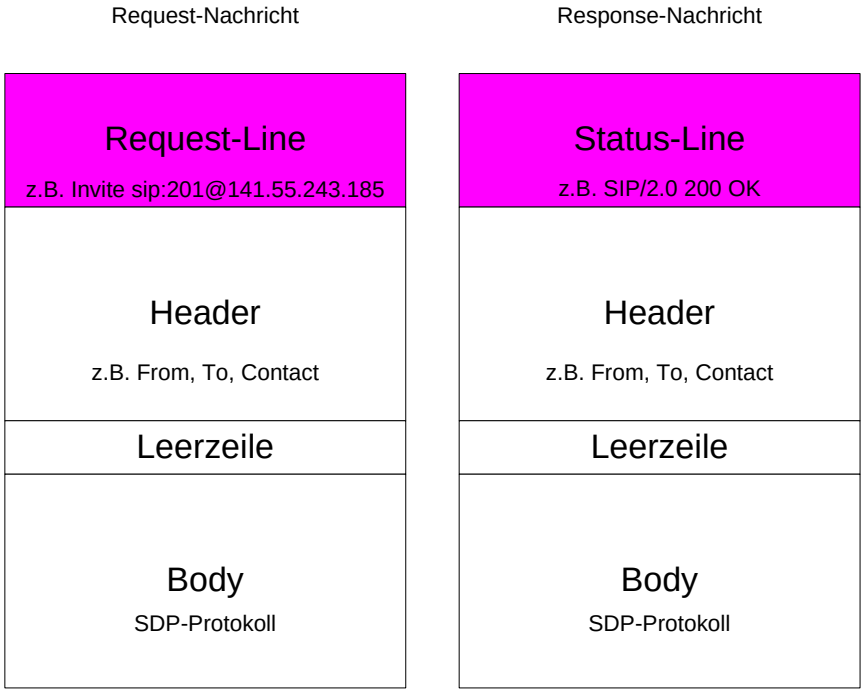

Abbildung 6: Request- und Response-Nachricht

Die Request-Line der Request-Nachricht beinhaltet neben der URI auch die Methode, die zum Einsatz kommt. In der Abbildung 6 ist als Beispiel Invite angegeben. Andere Request-Nachrichten sind z.B. Register oder ACK. Die Status-Line der Response-Nachricht enthält die SIP-Version und den Nachrichtentyp. Nachrichtentypen sind z.B. 200 OK oder 180

Ringing. Durch den Request-Header ist es möglich, Informationen des Client an den Server zu senden. Im folgenden Beispiel wird der Header einer Invite-Nachricht, mit der zu einer Sitzung eingeladen wird, dargestellt:

INVITE sip:1404@141.55.243.185 SIP/2.0
Via: SIP/2.0/UDP 141.55.243.185
To: 1404@141.55.243.185
From: Hello World!@141.55.243.185
Call-ID: 123456789@141.55.243.185
CSeq: 314159 INVITE
Contact: sip:Hello World!@141.55.243.185
Content-Type: application/sdp
Content-Length: 150

Im weiteren soll die Bedeutung der einzelnen Header-Felder erklärt werden [VoIP-info, 2010].

- Im ersten Feld wird die Request-Nachricht angegeben. Für das Beispiel ist das die Nachricht „Invite“, welche mit Adresse und SIP-Version angegeben wird.
- Im Feld „Via“ wird das Protokoll, die Version und die IP-Adresse für die Rückantwort des Request hinterlegt.
- Im Feld „To“ wird die SIP-Adresse des Empfängers eingetragen.
- Im Feld „From“ wird die SIP-Adresse des Senders eingetragen. Dieser Eintrag erscheint in der Rufnummernanzeige des Endgerätes.
- Das Feld „Call-ID“ definiert eine Zufallszeichenkette als eindeutige Nummer für das Gespräch.
- Im Feld „CSeq“ wird die Sequenznummer definiert. Diese dient dazu, dass auf der Partnerseite die richtige Reihenfolge der Nachrichten erkannt wird.
- Das Feld „Contact“ definiert die SIP-Adresse des Empfängers für die direkte Kommunikation.
- Durch das Feld „Content-Type“ wird das Format der Nutzdaten definiert. Für das Beispiel ist SDP eingetragen.
- Im Feld „Content-Length“ wird die Länge der Nutzdaten definiert (in Byte). Dadurch wird die Größe des Nachrichten-Bodys festgelegt.

Nach dem Header schließt sich, durch eine Leerzeile getrennt, der Body an. Der Body enthält die Nutzdaten bzw. die Multimediadaten.

-SDP

Durch das Session Description Protocol (SDP) wird eine Multimediasitzung beschrieben. Das Session Description Protocol wird im RFC 4566 definiert. [IETF, 2010]. Es werden Details für die Audio- und Video-Übertragung von den Teilnehmern ausgehandelt. Wie ein SDP-Body aufgebaut ist soll das folgende Beispiel zeigen:

v=0	Protokollversion
o=1234 1234 IN IP4 141.55.243.75	Erzeuger und Sitzungs-Identifizierung
s=SIP-Call	Name der Verbindung
m=audio 49972 RTP/AVP 80	Medienname und Adresse für Transport (Port)
c=IN IP4 141.55.243.75	optionale Verbindungsinformation
t=0 0	Zeitgrenze (Anfang bis Ende)
a=rtpmap:8 PCMA/8000	optionale Medienbeschreibung

Die einzelnen Felder werden nachfolgend erläutert.

- Im ersten Feld wird die Protokollversion eingetragen. Derzeit wird bei SDP die Version 0 verwendet.
- Im zweiten Feld wird eingetragen, wer die Sitzung erzeugt und wodurch sie identifiziert wird. Als Netztyp ist IN eingetragen. IN steht dabei für Internet. Daran schließt sich der Adresstyp, im Beispiel IP4 an. Das Feld wird mit der Adresse abgeschlossen.
- Im dritten Feld wird der Verbindung ein Name zugewiesen. Für das o.g. Beispiel ist der Name SIP-Call gewählt worden.
- Das vierte Feld dient der Medienbeschreibung. Das Beispiel bezieht sich auf eine Audio-Übertragung mit der Portnummer 49972. Daran schließt sich das verwendete Transportprotokoll an. In diesem Fall RTP/AVP 80, also das Real Time Protocol für Audio und Video mit dem Medienformat 80.
- Das fünfte Feld ist optional und gibt Informationen über die Verbindung. Einzutragen sind Netztyp, Adresstyp und Verbindungsadresse.
- Im sechsten Feld wird die Zeitgrenze festgelegt. Es wird die Startzeit und die Stopzeit eingetragen. Bei VoIP wird für beides üblicherweise 0 eingetragen.
- Das letzte Feld ist eine optionale Medienbeschreibung. Die Angaben im o.g. Beispiel beziehen sich auf die RTP-Angaben im m-Feld. Weiterhin ist festgelegt, mit welchem Verfahren kodiert werden soll und die dafür notwendige Taktrate.

3 Funktionaltest

Als Vorbereitung für die Erstellung des Praktikumsversuches wurde zunächst ein Funktionaltest mit Hilfe der IPS-Software als Eigenversuch durchgeführt. Der Umgang mit der IPS-Software sollte dadurch trainiert werden.

3.1 Allgemeines

Mit dem Funktionaltest sollen nunmehr die vorher festgelegten Funktionen eines Systems überprüft werden. Um die Arbeitsweise eines solchen Tests anschaulich zu machen, wurde ein Testsystem festgelegt. Der Test wurde an einem SIP-Telefon durchgeführt bzw. bei einem SIP-Telefon wurden ausgewählte Funktionalitäten getestet.

3.2 Testaufbau

Abbildung 7 veranschaulicht vereinfacht die Versuchsanordnung für den Testversuch. Als Testobjekt wird ein SIP-Telefon verwendet. Bei diesem Endgerät handelt sich um ein snom 190. Ein weiterer Bestandteil ist der PC mit der entsprechenden Testsoftware. Als Testsoftware wird IPS verwendet. Die Adressierung des Endgerätes erfolgt über die entsprechende IP-Adresse des Telefons. Der PC, auf dem die Testsoftware läuft hat eine eigene IP-Adresse.

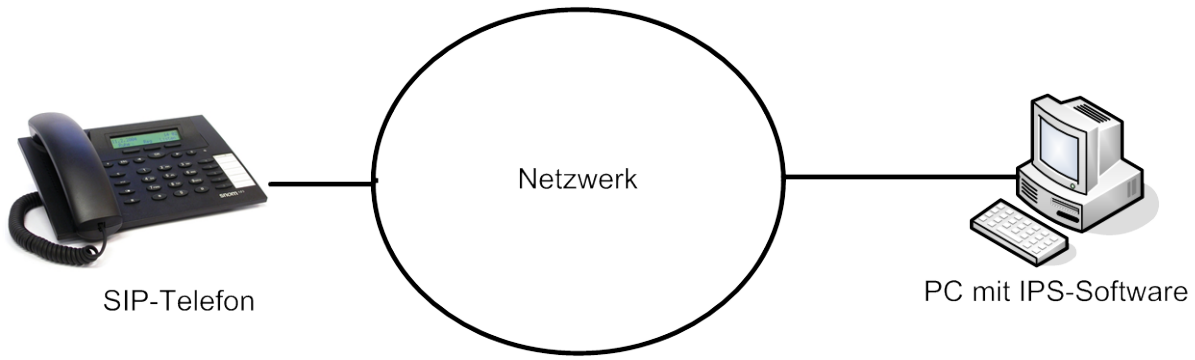

Abbildung 7: Versuchsanordnung

Sowohl der PC mit der Testsoftware als auch das SIP-Telefon sind an ein Netzwerk angeschlossen. Verwendet wird das Netzwerk der Hochschule Mittweida.

3.3 Testvorbereitung

Es ist ein Verbindungsaufbau und ein Verbindungsabbau zwischen dem IPS- Simulationssystem und dem SIP-Telefon zu realisieren. Der Simulator ist dabei der rufende Teilnehmer (A-Teilnehmer) und das SIP-Telefon der gerufene Teilnehmer (B-Teilnehmer). Gegenwärtig befindet sich die Verbindung im Zustand „Getrennt“. Mit dem Gesprächsaufbau wechselt das System in den Zustand „Wählend“. Dieser Zustand wird bis zum Ende des Gesprächsaufbaus beibehalten. Falls der Gesprächsaufbau erfolgreich war, befindet sich das System im Zustand „Verbunden“.

Ein erfolgreicher Gesprächsaufbau beginnt stets mit der Einladung zur Sitzung dem Invite durch den Simulator. Die Nachricht Invite enthält die Adresse des gerufenen Teilnehmers. Der gerufene Teilnehmer soll mit den Nachrichten Trying, Ringing und OK antworten. Voraussetzung dafür ist, dass der Hörer aufliegt. Ist das der Fall, so soll der Simulator ein ACK senden. Durch das Abheben des Hörers beim gerufenen Teilnehmer wird die Media Session eingeleitet. Ein Auflegen des Hörers führt zum vollständigen Abbruch des Gesprächs.

Durch den Test soll die Funktion, dass eine Verbindung zustande kommt nachgewiesen werden. Weiterhin wird geprüft, ob das gerufene SIP-Telefon klingelt, die Rufnummernanzeige und der Sprachkanal funktionieren. Der Sprachkanal wird dadurch getestet, indem der Simulator einen 1kHz-Ton sendet, welcher im Hörer des B-Teilnehmers zu hören ist.

Der Auf- und Abbau der Verbindung wird durch das Session Initiation Protocol (SIP) geregelt. Das Protokoll wird in RFC 3261 spezifiziert.
Durch den Simulator werden die einzelnen Anfragen gesendet, und es wird auf entsprechende Reaktionen gewartet. Sollte eine unerwartete oder falsche Nachricht an den Simulator zurückgeschickt werden, so wird im Trace ein Error angezeigt.
Die Simulation wird durch das Simulationsprogramm mit der Request-Methode Invite eingeleitet. Die Abläufe werden durch das Sequenzdiagramm (Abbildung 8) verdeutlicht. Im Weiteren wir darauf näher eingegangen.

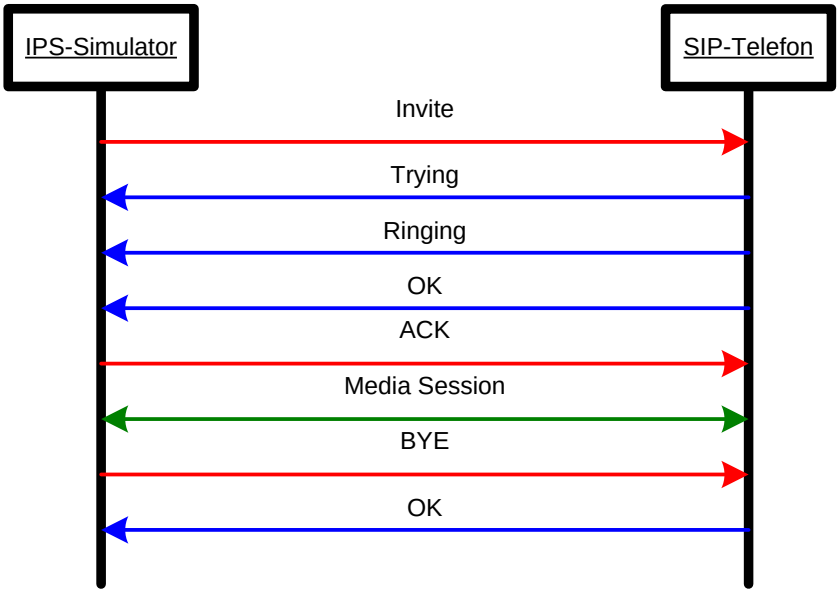

Abbildung 8: Sequenzdiagramm Funktionaltest

Das Sequenzdiagramm zeigt den IPS-Simulator und auf der Gegenseite das SIP-Telefon. Durch die Pfeile wird die Richtung angegeben, in die die jeweilige Nachricht gesendet wird. Nachdem der Simulator das Invite an das Telefon gesendet hat, erwartet er die Nachrichten Trying, Ringing und OK. Erhält der Simulator diese Nachrichten, quittiert er dies dem Telefon mit ACK. Nach erfolgreichem Verbindungsaufbau folgt die Media Session. Wird die Verbindung beendet schickt der Simulator dem Telefon die Nachricht BYE und erwartet darauf die Nachricht OK.

3.3.1 Die Cotrol-Datei

Als erster Schritt für den Funktionaltest muss eine Control-Datei erstellt werden. In der Abbildung 9 wird der Quelltext der vollständigen Control-Datei abgebildet. Die Control-Datei enthält einige Informationen (Abbildung 9). Als erstes wird die Version festgelegt. Für den hier durchgeführten Funktionaltest ist das die Version 1.1. Der Praktikumsversuch arbeitet dann mit der Version 2.0. Als zweites wird angegeben, dass es sich um eine Control-Datei handelt. Es wird ein Testblock erzeugt, in dem die Größe des Ressourcenbereiches festgelegt wird. Es können Werte von 0 bis 127 eingestellt werden. Als nächstes wird der Time-Typ festgelegt. Es kann zwischen CycleTime, IdleTime und Load gewählt werden. Für den Funktionaltest wird die IdleTime mit einem Wert von 1000 verwendet. Das entspricht einer Zeit von einer Sekunde. Die nächste Einstellung, die vorgenommen werden muss, ist die Anzahl der Wiederholungen. Für den Funktionaltest wird für Loop „eins“ eingestellt. Das bedeutet

der Testdurchlauf wird einmal ausgeführt. Als nächstes wird der Event-Typ ausgewählt. In diesem Zusammenhang wird die Startbedingung für das Test-Case festgelegt. Zum Schluss wird der Testblock beendet und die Control-Datei wird gespeichert.

```
SslVersion 1.1;            /*Angabe der Version*/

SimulatorControlFile;      /*Das ist eine Control-Datei*/

TestBlock                  /*Anfang des Testblocks*/
  Resources = 0;           /*Resourcenbereich für den Testblock*/
  IdleTime  = 1000;        /*Wartezeit zwischen zwei Anrufen in Millisekunden*/
  Loop = 1;                /*Anzahl der Wiederholungen bzw. Durchläufe*/
  On WakeupTimer           /*Timer der Control-Datei, angabe des Event-Typs*/
    Start Client;          /*Startet ein Test-Case*/
EndTestBlock;              /*Ende des Testblocks*/
```

Abbildung 9: Quelltext der Conrtol-Datei

3.3.2 Die Script-Datei

Der zweite Teil, aus dem ein Testfall besteht, ist die Script-Datei. In ihr ist der Ablauf des Simulationstests beschrieben. Durch die Script-Datei wird unter anderem das Session Initiation Protocol beschrieben und zur Ausführung gebracht. Es werden Reaktionszeiten festgelegt. Durch diese Reaktionszeiten wird die Wartezeit, z.B. auf eine Antwort, eingeschränkt. Die vollständige Script-Datei zum Test ist im Anlagenteil unter 6.1 hinterlegt.

3.4 Testdurchführung

Voraussetzung für die Durchführung des Simulationstests ist, dass die Control-Datei sowie die Script-Datei vollständig und korrekt sind. Bevor die Simulation startet, ist es erforderlich, die Dateien zu kompilieren. Die Simulation wird gestartet und damit das Testprogramm ausgeführt.
Es wurden verschiedene Szenarien durchgespielt und untersucht, und zwar:

- der normale Zustand des Telefons (Hörer ist aufgelegt),
- der Hörer ist zu Beginn des Tests bereits abgehoben und
- das Klingeln des Telefons wird ignoriert.

Weiterhin werden verschiedene Texte, Zeichen bzw. Ziffern in der Rufnummernanzeige angezeigt.

3.5 Auswertung

Die Überprüfung des SIP-Telefons wurde erfolgreich abgeschlossen. In dem Test, in dem der normale Zustand des Telefons (Hörer ist aufgelegt), geprüft wurde, stimmt das Sequenzdiagramm im Trace mit dem Sequenzdiagramm (Abbildung 8) überein. Es sind keine Fehler aufgetreten. Dieses Ergebnis war zu erwarten, da es sich um ein funktionstüchtiges Telefon handelte. Interessant ist, wie die Rufnummernanzeige durch Zahlen bzw. Text beeinflusst werden kann. Dadurch ist es möglich, diese auf korrekte Funktion zu überprüfen. Bei bereits abgehobenem Hörer zu Beginn des Tests, bekam das Testsystem eine unerwartete Nachricht und im Trace wurde ein Error ausgegeben.
Abschließend ist festzustellen, dass der als Eigenversuch durchgeführte Funktionaltest sich als brauchbare Testmethode erwiesen hat und für Übungszwecke in Form eines Praktikumsversuches in der studentischen Ausbildung geeignet ist
Die Anleitung dafür wird mit dem folgenden Gliederungspunkt 4 „Praktikumsversuch zum Thema Testen“ gegeben.

4 Praktikumsversuch zum Thema Testen

Das Kapitel 4 beinhaltet die vollständige Praktikumsanleitung für den von den Studenten auszuführenden Praktikumsversuch. Dadurch kommt es in einigen Teilen z.T. zu Wiederholungen von Themen, die in den vorhergehenden Kapiteln bereits ausführlich erörtert werden.

4.1 Einleitung zum Versuch

Versuch für Studienrichtung:

- Informations- und Elektrotechnik

Versuchsziel:

- Schreiben eines Testprogramms, mit dem die Funktionalität der Software-Vermittlungsanlage Asterisk getestet werden soll

Inhalte:

- Vorgehensmodelle
- Kenntnisgewinn über die verschiedenen Softwareentwurfsverfahren
- Überblick verschaffen über verschiedene Tests
- Versuch zum Funktionaltest

Vorkenntnisse:

- Kenntnisse aus der Vorlesung VoIP
- Kenntnisse aus der Vorlesung Softwarelebenszyklus

Notwendige Hardware/Software:

- Asterisk-Server
- PC mit vorinstallierter IPS-Software
- SIP-Telefon
- ISDN-Telefon

Aufgabenstellung zum Versuch:

Schreiben Sie mit Hilfe der IPS-Software ein Testprogramm, mit dem, unter Beachtung des SIP-Protokolls, die Registrierung am Asterisk-Server, eine Verbindung zu einem SIP-Telefon und eine Verbindung zu einem ISDN-Telefon getestet werden können. Die Verbindung zum SIP-Telefon soll über den Asterisk-Server erfolgen. Die Verbindung zum ISDN-Telefon soll ebenfalls über den Astrisk-Server und weiter über die Hipath hergestellt werden.
Die Vorgehensweise, wie die Testfälle programmiert werden, wird in den Punkten 4.3, 4.4 und 4.5 schrittweise erklärt.

Versuchsaufbau:

Die Abbildung 10 zeigt ausgehend von der Aufgabenstellung den prinzipiellen Versuchsaufbau des zu testenden Kommunikationssystems. Am Netzwerk der Hochschule sind der Asterisk-Server und der PC mit dem IPS-Simulator angeschlossen.

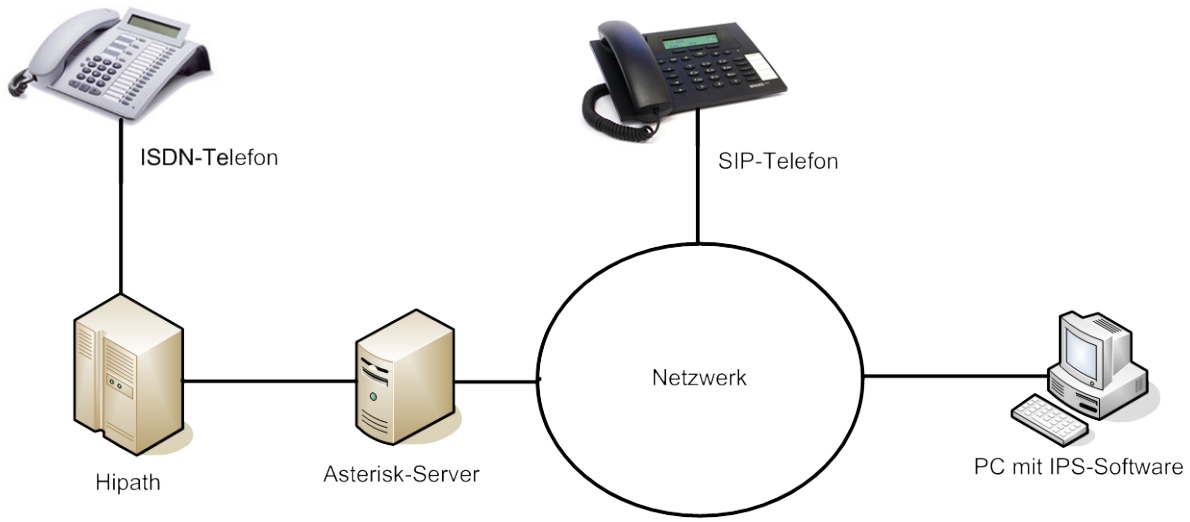

Abbildung 10: Versuchsaufbau

4.2 Grundlagen

In den Grundlagen soll kurz erläutert werden, was für Schritte notwendig sind, um neue Produkte zu entwickeln. Verschiedene Entwicklungsmodelle sollen erklärt werden und welche Testmöglichkeiten und Verfahren existieren.

Die Entwicklung neuer Produkte ist in der heutigen Zeit eine umfangreiche, aufwendige und zeitintensive Aufgabe. Eine große Rolle spielt dabei der Kunde. Er stellt die Anforderungen und erwartet ein Produkt, das seinen Vorstellungen gerecht wird.
Es existieren verschiedene Vorgehensmodelle, welche den Entwicklungsprozess erleichtern sollen. In diesen Vorgehensmodellen sind sämtliche Arbeitsabläufe geregelt. Die Vorgehensmodelle werden nach ihrem Projektumfang und dem Kundenbezug eingeteilt. Bezogen auf den Projektumfang bedeutet das, wie flexibel ist die Arbeit, wie hoch ist der Personalbedarf oder der Zeitaufwand. Mit dem Kundenbezug wird ausgedrückt, wie stark beispielsweise spätere Änderungen berücksichtigt werden oder wie sehr der Kunde am Produkt mitarbeitet.
Als Beispiel soll das Wasserfallmodell kurz erläutert werden. Es ist ein klassisches Entwicklungsmodell. Vielen anderen Modellen liegt dieses Modell zugrunde. Die Abbildung 11 zeigt das Wasserfallmodell mit seinen Phasen.

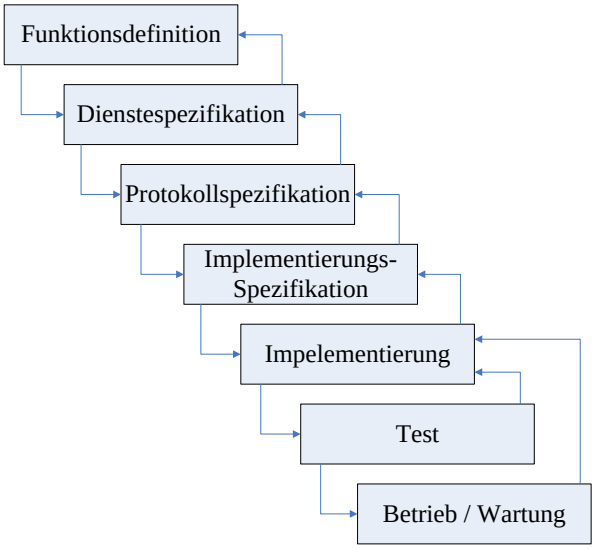

Abbildung 11: Wasserfallmodell

Die Phasen haben zum Inhalt:

- Funktionsdefinition: Eine meist verbale Beschreibung der beabsichtigten Funktion.
- Dienstespezifikation: Eine Beschreibung, welche den Aufbau und das Verhalten des Systems bezogen auf die Dienste beschreibt.
- Protokollspezifikation: Eine Beschreibung, die den Aufbau und das Verhalten des Systems bezogen auf die Protokolle beschreibt.
- Implementierungsspezifikation: Ist eine implementierungsabhängige Beschreibung der Funktionalität eines Protokolls.
- Implementierung: Ist die Realisierung der beabsichtigten Funktionalität in einem System.

- Test: Ist die Überprüfung auf Vollständigkeit und Korrektheit einer Implementation.
- Betrieb/Wartung: Ist der Einsatz des Systems beim Kunden und die Betreuung des Systems vor Ort.

Der Entwicklungsprozess beim Wasserfallmodell ist in Phasen aufgeteilt. Bevor mit einer neuen Phase begonnen wird, muss die vorherige Phase abgeschlossen sein. Dadurch ist das Wasserfallmodell nicht sehr flexibel. Die Anforderungen müssen zu einem frühen Zeitpunkt festgelegt sein und können kaum an geänderte Wünsche angepasst werden.
Weitere Entwicklungsmodelle sind das V-Modell, das Prototypen-Modell und die Agile Softwareentwicklung. Eine Testphase haben alle diese Modelle. Mit der Durchführung von Tests sollen Fehler so früh wie möglich erkannt werden, um diese so schnell wie möglich zu beseitigen. Zu diesem Zweck gibt es verschiedene Techniken, einen Test durchzuführen.
Es bestehen mehrere Möglichkeiten die Testtechniken zu klassifizieren. Eine Möglichkeit ist die Klassifikation nach Whiteboxtests und Blackboxtests. Beim Whiteboxtest wird die Untersuchung direkt am Quellcode vorgenommen. Beim Blackboxtest ist der Quellcode nicht bekannt bzw. besteht nicht die Möglichkeit, an ihm zu testen. Beim Blackboxtest wird geprüft ob, das System mit den Spezifikationen übereinstimmt. Die Steuerung des Testablaufes ist nur von außen durch entsprechende Wahl der Eingangsdaten möglich.
Eine andere Möglichkeit ist die Klassifizierung in funktionale- und nichtfunktionale Tests. Die funktionalen Tests betrachten die korrekte Funktion eines Systems. Ein Beispiel für einen solchen Test ist der Funktionaltest, welcher im Praktikumsversuch zur Anwendung kommt. Es sollen im Versuch verschiedene Funktionen getestet werden, wie z.B. das Klingeln eines Telefons. Ein Vertreter für einen nichtfunktionalen Test ist der Performancetest (Leistungstest). Mit diesem Test wird untersucht, wie sich ein System verhält, das viele Verbindungen gleichzeitig ausführen soll. Weiterhin wird dabei das Zeitverhalten betrachtet.

Voice over IP

Voice over IP ist Sprachübertragung über Rechnernetzwerke. Es wird ein Netzwerk für mehrere Dienste genutzt. Einer dieser Dienste ist die Telefonie. Im Gegensatz zur Telefonie über das Fernsprechnetz wird die Nachricht bei VoIP paketvermittelt. Die Standardisierung erfolgt hauptsächlich durch die ITU-T, die IETF und das ETSI. Die drei Organisationen haben unabhängig voneinander eigene Standards entwickelt. Die Möglichkeiten der Teilnehmeradressierung sind vielfältig, z.B. *sip:meier@htwm.de*, *sip:+49-3727-581404@htwm.de* oder auch *sip:meier@141.55.xxx.xxx*.
Eine Kommunikationsbeziehung wird in drei Phasen unterteilt:

- Sitzungsaufbau
- Nutzung

- Sitzungsabbau

Der Sitzungsauf und –abbau wird durch SIP geregelt. SIP steht für Session Initiation Protocol, und es handelt sich dabei um ein Signalisierungsprotokoll. Das von der IETF entwickelte Session Initiation Protocol wird im RFC 3261 definiert. [IETF, 2009]. Die Kommunikation bei SIP erfolgt über Nachrichten (Messages). Diese Nachrichten werden zwischen den Clients und Servern ausgetauscht. Es werden zwei Arten von Nachrichten unterschieden: Requests (Anfrage, Anforderung) und Responses (Antworten). Eine Request-Nachricht besteht aus einer Request-Line dem Request-Header und dem Body. Im folgenden Beispiel wird der Header eine Invite-Nachricht, mit der zu einer Sitzung eingeladen wird, dargestellt:

INVITE sip:1404@141.55.243.185 SIP/2.0
Via: SIP/2.0/UDP 141.55.243.185
To: 1404@141.55.243.185
From: Hello World!@141.55.243.185
Call-ID: 123456789@141.55.243.185
CSeq: 314159 INVITE
Contact: sip:Hello World!@141.55.243.185
Content-Type: application/sdp
Content-Length: 150

Im weiteren soll die Bedeutung der einzelnen Header-Felder erklärt werden [VoIP-info, 2010].

- Im ersten Feld wird die Request-Nachricht angegeben. Für das Beispiel ist das die Nachricht „Invite“, welche mit Adresse und SIP-Version angegeben wird.
- Im Feld „Via“ wird das Protokoll, die Version und die IP-Adresse für die Rückantwort des Request hinterlegt.
- Im Feld „To“ wird die SIP-Adresse des Empfängers eingetragen.
- Im Feld „From“ wird die SIP-Adresse des Senders eingetragen. Dieser Eintrag erscheint in der Rufnummernanzeige des Endgerätes.
- Das Feld „Call-ID“ definiert eine Zufallszeichenkette als eindeutige Nummer für das Gespräch.
- Im Feld „CSeq“ wird die Sequenznummer definiert. Diese dient dazu, dass auf der Partnerseite die richtige Reihenfolge der Nachrichten erkannt wird.
- Das Feld „Contact“ definiert die SIP-Adresse des Empfängers für die direkte Kommunikation.

- Durch das Feld „Content-Type“ wird das Format der Nutzdaten definiert. Für das Beispiel ist SDP eingetragen.
- Im Feld „Content-Length“ wird die Länge der Nutzdaten definiert (in Byte). Dadurch wird die Größe des Nachrichten-Bodys festgelegt.

Nach dem Header schließt sich, durch eine Leerzeile getrennt, der Body an. Der Body enthält die Nutzdaten bzw. die Multimediadaten, die durch SDP beschrieben werden.
Durch das Session Description Protocol (SDP) wird eine Multimediasitzung beschrieben. Das Session Description Protocol wird im RFC 4566 definiert. [IETF, 2010]. Es werden Details für die Audio- und Video-Übertragung von den Teilnehmern ausgehandelt. Wie ein SDP-Body aufgebaut ist soll das folgende Beispiel zeigen:

v=0	Protokollversion
o=1234 1234 IN IP4 141.55.243.75	Erzeuger und Sitzungs-Identifizierung
s=SIP-Call	Name der Verbindung
m=audio 49972 RTP/AVP 80	Medienname und Adresse für Transport (Port)
c=IN IP4 141.55.243.75	optionale Verbindungsinformation
t=0 0	Zeitgrenze (Anfang bis Ende)
a=rtpmap:8 PCMA/8000	optionale Medienbeschreibung

Die einzelnen Felder werden nachfolgend erläutert.

- Im ersten Feld wird die Protokollversion eingetragen. Derzeit wird bei SDP die Version 0 verwendet.
- Im zweiten Feld wird eingetragen, wer die Sitzung erzeugt und wodurch sie identifiziert wird. Als Netztyp ist IN eingetragen. IN steht dabei für Internet. Daran schließt sich der Adresstyp, im Beispiel IP4 an. Das Feld wird mit der Adresse abgeschlossen.
- Im dritten Feld wird der Verbindung ein Name zugewiesen. Für das o.g. Beispiel ist der Name SIP-Call gewählt worden.
- Das vierte Feld dient der Medienbeschreibung. Das Beispiel bezieht sich auf eine Audio-Übertragung mit der Portnummer 49972. Daran schließt sich das verwendete Transportprotokoll an. In diesem Fall RTP/AVP 80, also das Real Time Protocol für Audio und Video mit dem Medienformat 80.
- Das fünfte Feld ist optional und gibt Informationen über die Verbindung. Einzutragen sind Netztyp, Adresstyp und Verbindungsadresse.

- Im sechsten Feld wird die Zeitgrenze festgelegt. Es wird die Startzeit und die Stopzeit eingetragen. Bei VoIP wird für beides üblicherweise 0 eingetragen.
- Das letzte Feld ist eine optionale Medienbeschreibung. Die Angaben im o.g. Beispiel beziehen sich auf die RTP-Angaben im m-Feld. Weiterhin ist festgelegt, mit welchem Verfahren kodiert werden soll und die dafür notwendige Taktrate.

4.3 1. Versuchsaufgabe: Registrierung

Versuchsdurchführung:
Öffnen Sie den Ordner mit dem vorbereiteten IPS-Programm. Der Ordner enthält folgende Dateien:

- test.cfg
- test.ctl
- test.scr

Ein Testfall besteht aus einer Script-Datei, einer Control-Datei und einer Konfigurationsdatei. Die Control-Datei und die Konfigurationsdatei sind für den Test fertig vorbereitet.

Der Quellcode für das Testprogramm wird in die Script-Datei geschrieben. Als erster Test soll die Registrierung am Asterisk-Server getestet werden. Jedes Endgerät, welches an Asterisk angeschlossen wird, muss eine Registrierungsprozedur durchführen. Die Abbildung 12 verdeutlicht den Ablauf der Registrierung.

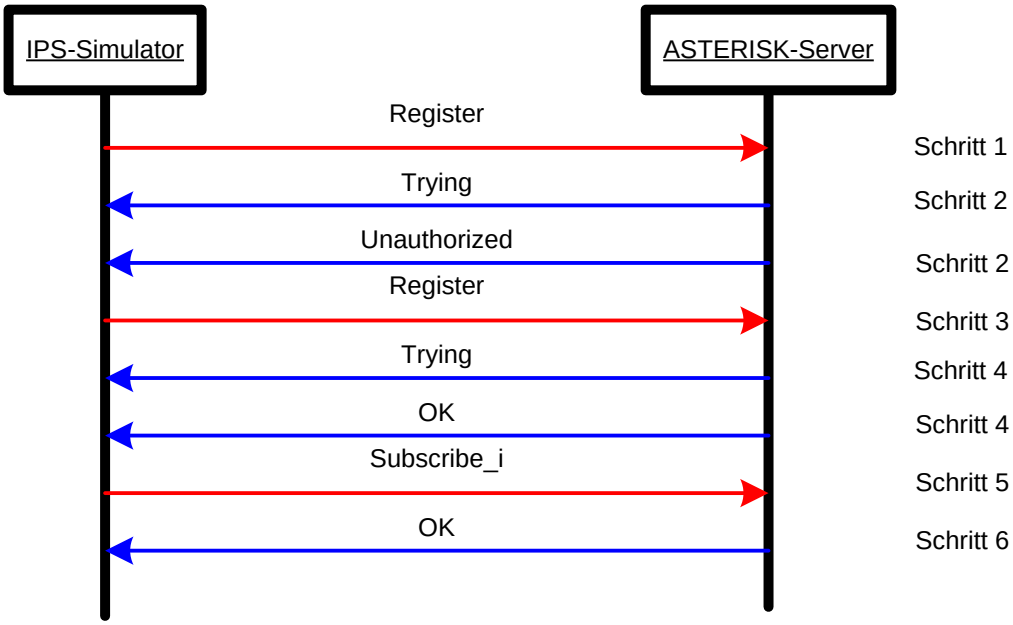

Abbildung 12: Sequenzdiagramm Registrierung

Man unterscheidet zwischen Nachrichten (Messages), die gesendet werden und Nachrichten, welche empfangen werden. Diese Messages sind in einem Pool vorbereitet und müssen diesem nach Bedarf entnommen werden. Diesen Pool findet man in der Menüleiste unter Script im Punkt Command. Folgende Schritte sind zu gehen:

Schritt 1: Register

Entsprechend dem Sequenzdiagramm wird die erste Nachricht „Register“ an den Server gesendet. Die Nachricht enthält eine Requestline, in der die Request URI eingetragen wird und einen Reqeuestheader mit Contact, To und From.
Als erstes muss die Request URI festgelegt werden. Als zweites wird der Header erstellt und die IP des Servers ist unter Conntact einzutragen. Weiterhin muss im Header unter To und From die jeweilige Teilnehmeradresse eingetragen werden. Die Bildung dieser Teilnehmeradresse wird durch ENUM festgelegt. Die Adresse beinhaltet den Dienst, URI und weitere Informationen. Es wird in der Menüleiste begonnen mit:

Script → Command → Put message into buffer [OK] → Modified message from pool [OK] → Register [OK] → Request [Sub/Set] → RequestLine [Sub/Set] → RequestURI [Sub/Set] → RequestURI [Sub/Set]

In der Eingabeaufforderung Value ist die IP-Adresse des Servers anzugeben (Abbildung 13) und mit OK zu bestätigen.

sip:141.55.243.185 [OK]

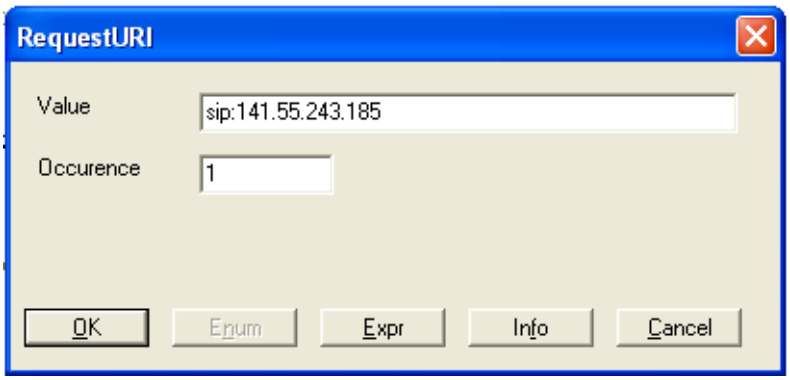

Abbildung 13: Request URI

In dem Fenster, dass sich dann öffnet, durch Anklicken des Button Cancel in das Fenster Request zurückkehren. Im Fenster Request beginnen mit:

RequestHeader [Sub/Set] → Contact [Insert] → [OK] → [OK]

Abbildung 14: Insert Parameter

Contact [Sub/Set] → Contact [Sub/Set] →

In der Eingabeaufforderung Value ist die IP-Adresse des Servers anzugeben (Abbildung 15) und mit OK zu bestätigen.

sip:@141.55.243.185 [OK]

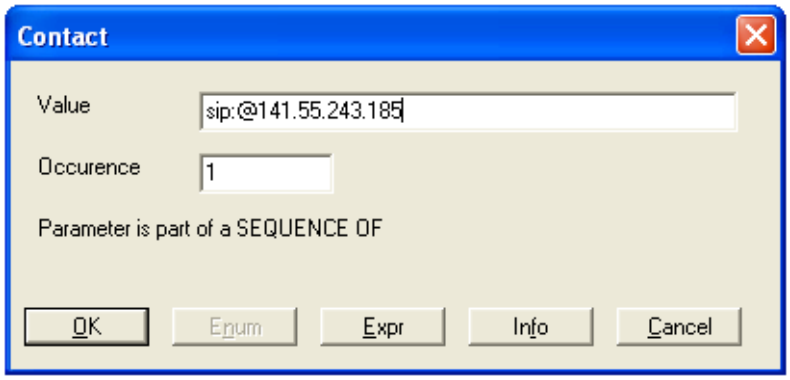

Abbildung 15: Contact

In dem Fenster, dass sich dann öffnet, durch Anklicken des Button Cancel in das Fenster RequestHeader zurückkehren. Im Fenster RequestHeader beginnen mit:

To [Insert] → [OK] → [OK]

Abbildung 16: Insert Parameter

To [Sub/Set] → To [Sub/Set] →
In der Eingabeaufforderung Value ist die Teilnehmeradresse des eigenen PC´s einzutragen (Abbildung 17) und mit OK zu bestätigen. z.B.:

sip:203@141.55.243.75 [OK]

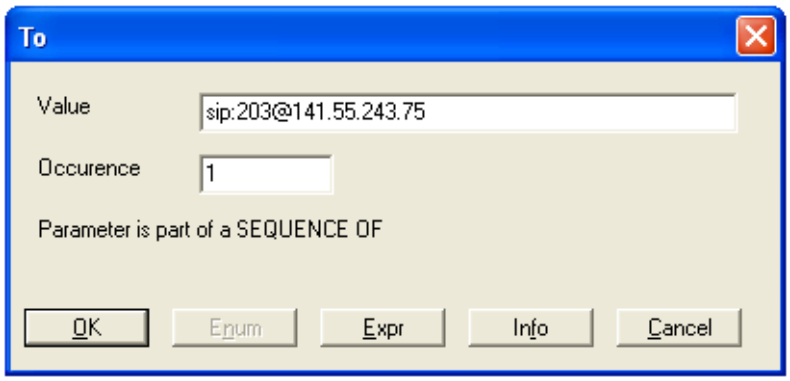

Abbildung 17: To

In dem Fenster, dass sich dann öffnet, durch Anklicken des Button Cancel in das Fenster RequestHeader zurückkehren. Im Fenster RequestHeader beginnen mit:

From [Insert] → [OK] → [OK]

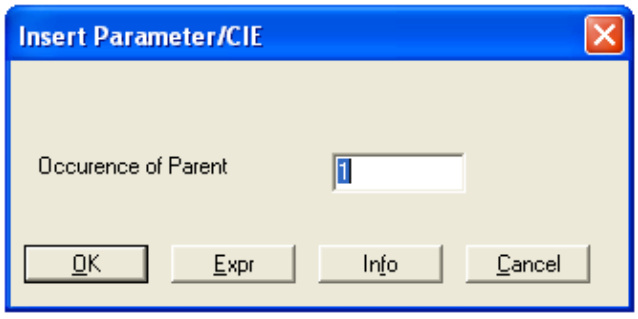

Abbildung 18: Insert Parameter

From [Sub/Set]→ From [Sub/Set]

In der Eingabeaufforderung Value ist die Teilnehmeradresse des eigenen PC´s einzugeben (Abbildung 19) und mit OK zu bestätigen. z.B.:

sip:203@141.55.243.75 [OK]

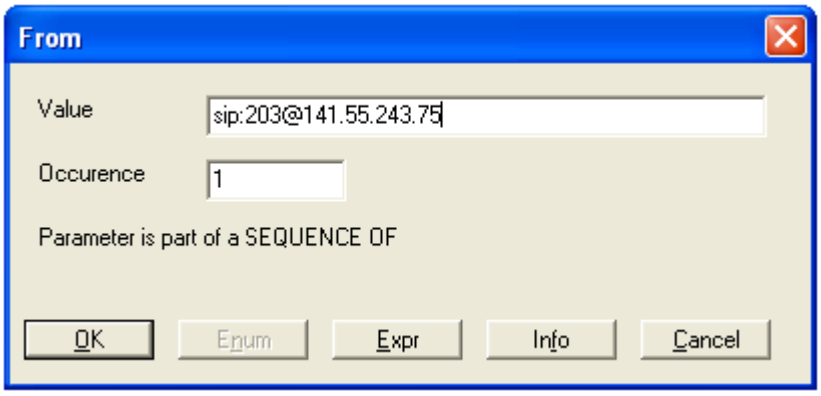

Abbildung 19: From

In dem Fenster, dass sich dann öffnet, ist durch Anklicken des Buttons Cancel die Auswahl zu verlassen.

Eine zu schickende Nachricht muss jeweils mit Send und Semikolon abgeschlossen werden. Die Abbildung 20 zeigt den Vollständigen Header für die Regisrierung.

```
Put(Modify(Pool("REGISTER", MSG_UNI_POOL))        /* |---- REGISTER    --->| */
        RequestURI := "sip:141.55.243.185";       //IP des Asterisk-Servers
        Insert c_P_RequestHeader.c_P_Contact;
        Contact := "sip:141.55.243.185";          // IP Asterisk-Server
        Insert c_P_RequestHeader.c_P_To;
        To := "sip:203@141.55.243.75";            //Nutzername und eigene IP
        Insert c_P_RequestHeader.c_P_From;
        From := "sip:203@141.55.243.75";          //Nutzername und eigene IP
    EndModify);

  Send;
```

Abbildung 20: Register-Header

Schritt 2: Trying und Unauthorized

Entsprechend dem Sequenzdiagramm werden die Nachrichten „R100 Trying“ und „R401 Unauthorized“ vom Server geschickt. Es wird in der Menüleiste begonnen mit:

Script → Command → Wait for event [OK] → Message [OK] → Trying (100) [OK]

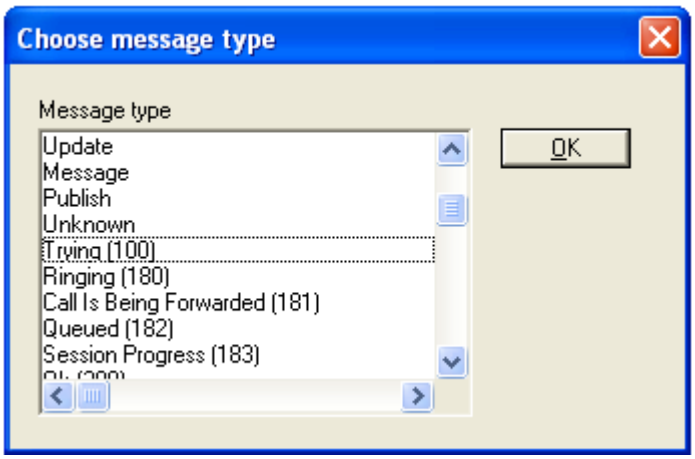

Abbildung 21: Messagetype (100)

Wait for event [OK] → Message [OK] → Unauthorized (401) [OK]

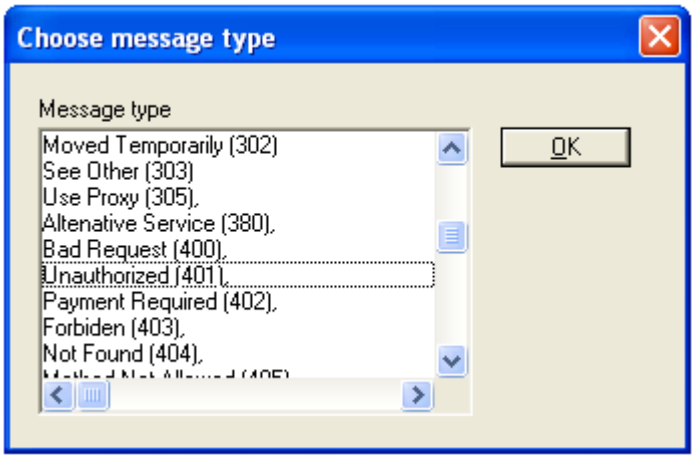

Abbildung 22: Messagetyp (401)

Schritt 3: Register wiederholen

Die Nachricht Register wird das zweitemal an den Server geschickt. Der Schritt 1 wird an dieser Stelle wiederholt.

Schritt 4: Trying und OK

Als nächstes werden die Nachrichten „R100 Trying“ und „R200 OK“ vom Server erwartet. Es wird in der Menüleiste begonnen mit:

Script → Command → Wait for event [OK] → Message [OK] → Trying (100) [OK]

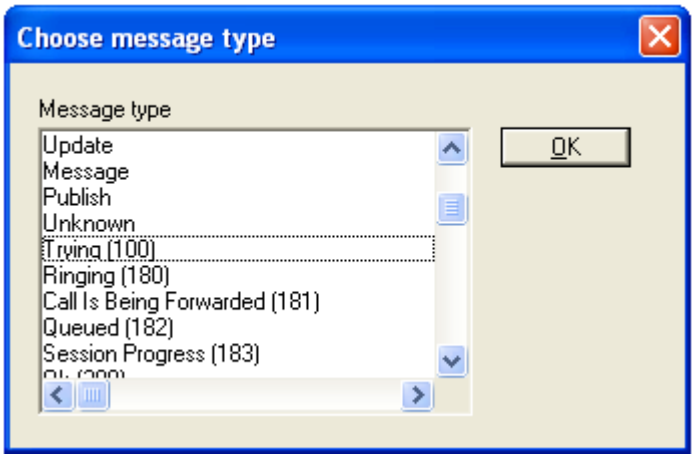

Abbildung 23: Messagetyp (100)

Wait for event [OK] → Message [OK] → OK (200) [OK]

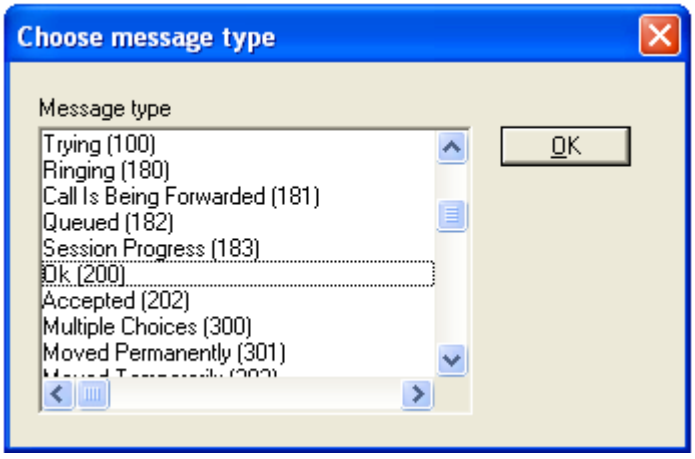

Abbildung 24: Messagetyp (200)

Schritt 5: Subscribe_I

Laut Sequenzdiagramm ist „Subscribe_I" die nächste Nachricht, welche an den Server geschickt wird. Begonnen wird in der Menüleiste mit:

Script → Command → Put message into buffer [OK] → Modified message from pool [OK] → Subscribe_I [OK] → Reqest [Sub/Set] → ReqestLine [Sub/Set] → RequestURI [Sub/Set] → RequestURI [Sub/Set]

In der Eingabeaufforderung Value ist die Adresse des eigenen PC´s anzugeben. z.B.:

sip:203@141.55.243.75 [OK]

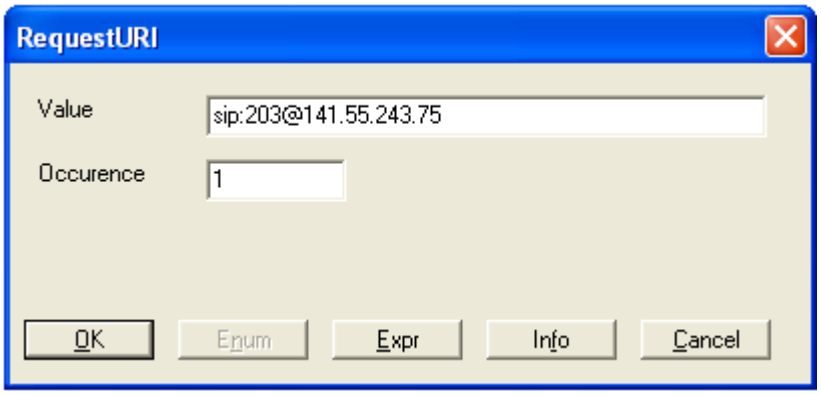

Abbildung 25: Request URI

Mit Cancel das Menü verlassen. Die Eingabe ist wieder mit „Send;“ abzuschließen.

Schritt 6: OK

Abschließend wird vom Server die Antwort „R200 OK“ geschickt. Siehe dazu Schritt 2 bzw. Schritt 4.

Schritt 7: Speichern

Nachdem die Struktur der Testsimulation fertig gestellt ist, muss diese gespeichert werden. Dazu wählt man in der Menüleiste:

File → Save

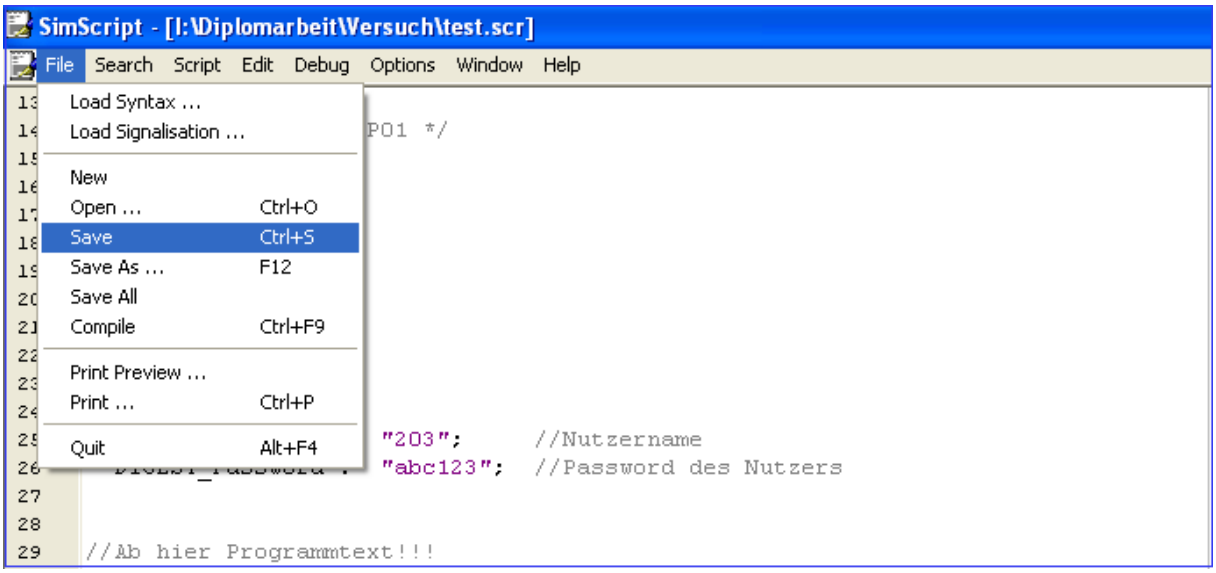

Abbildung 26: Speichern

Unter File besteht die Möglichkeit das Programm zu kompilieren.

Schritt 8: Kompilieren

Vor der Ausführung des Tests ist es zwingend erforderlich, das Programm in der Control-Datei zu kompilieren. Dazu muss die Control-Datei geöffnet werden. In der Menüleiste unter File ist der Befehl Compile zu finden (Abbildung 27).

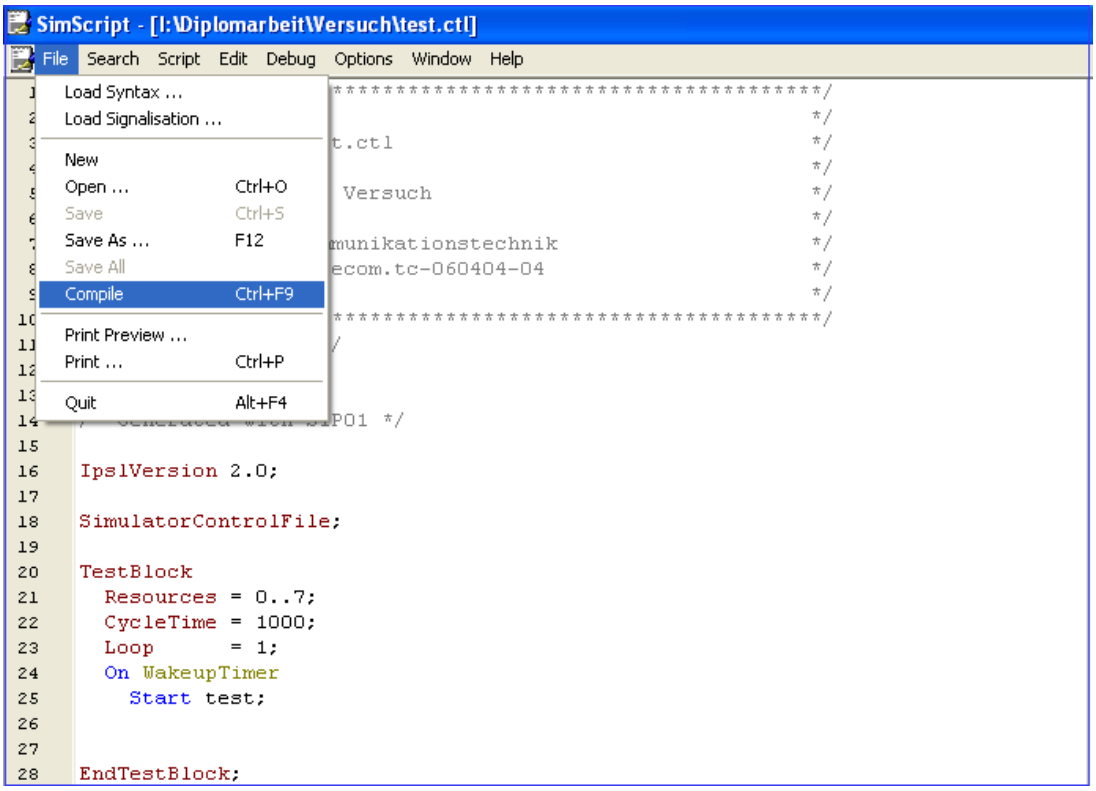

Abbildung 27: Kompilieren

Schritt 9: Testdurchführung

Nach erfolgreicher Kompilierung, kann der Test durchgeführt werden. Der Test wird automatisch durch das Öffnen der Konfigurationsdatei gestartet (Abbildung 28). Es öffnet sich das IPS-Fenster. Im IPS-Fenster wird im Trace-Teil der Testverlauf angezeigt. Bei erfolgreicher Registrierung entsprechen die Abläufe im Trace den Abläufen im Sequenzdiagramm.

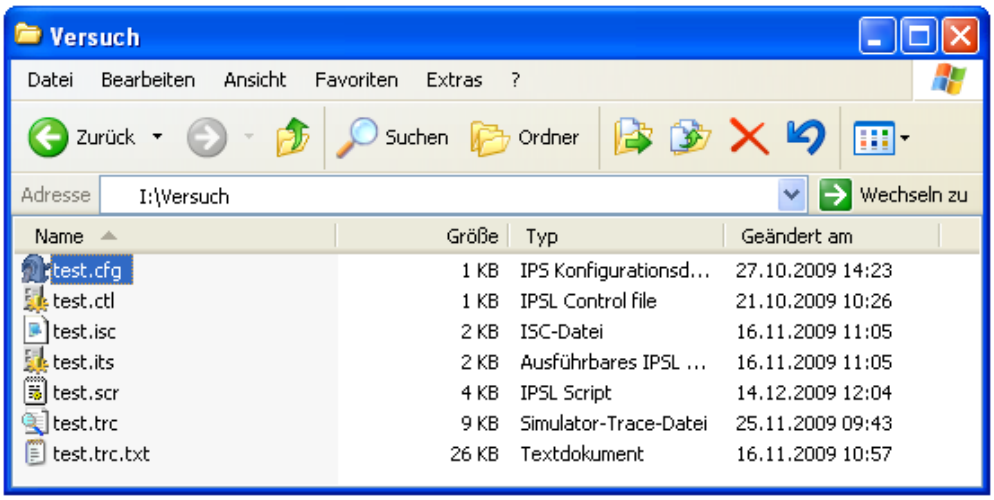

Abbildung 28: Teststart

4.4 2. Versuchsaufgabe: Verbindungsaufbau zu einem SIP-Telefon

Bei dieser Aufgabe soll die Verbindung zum Server und weiter zum SIP-Telefon eingeleitet und aufgebaut werden. Dazu wird das unter Punkt 4.3 erstellte Testprogramm zur Registrierung erweitert. Das Sequenzdiagramm in Abbildung 29 stellt den Ablauf dar.

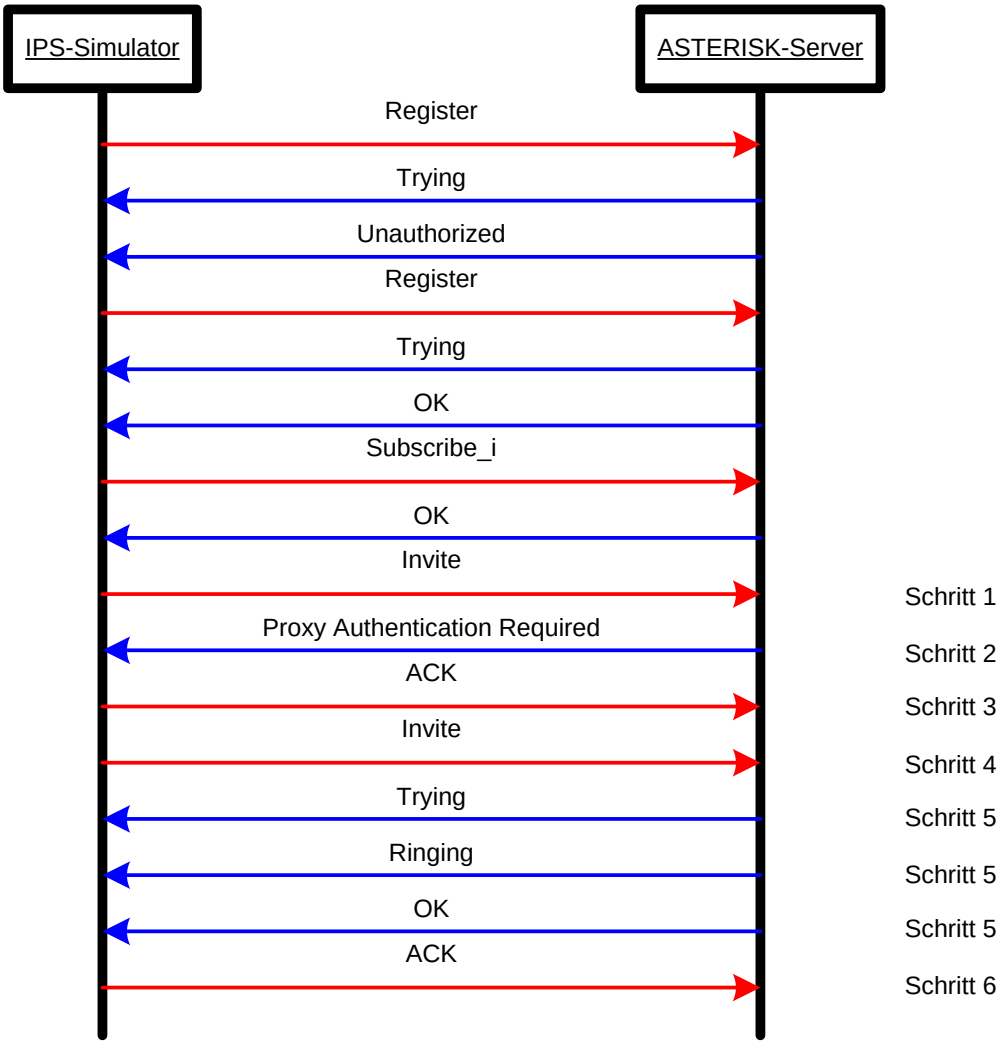

Abbildung 29: Sequenzdiagramm Verbindung zum SIP-Telefon

Folgende Schritte sind zu gehen:

Schritt 1: Invite

Die eigentliche Sitzung (Session) wird mit „Invite“ eingeleitet. Die Invite-Nachricht wird vom Simulator an den Asterisk-Server geschickt. Begonnen wird wieder in der Menüleiste mit:

Script → Command → Put message into buffer [OK] → Modified message from pool [OK] → Invite [OK] → Request [Sub/Set] → Requestline [Sub/Set] → RequestURI [Sub/Set] → RequestURI [Sub/Set]

In der Eingabeaufforderung Value sind die Rufnummer des zu rufenden Teilnehmers sowie die IP-Adresse des Asterisk-Servers einzutragen. z.B.:

sip:201@141.55.243.185 [OK]

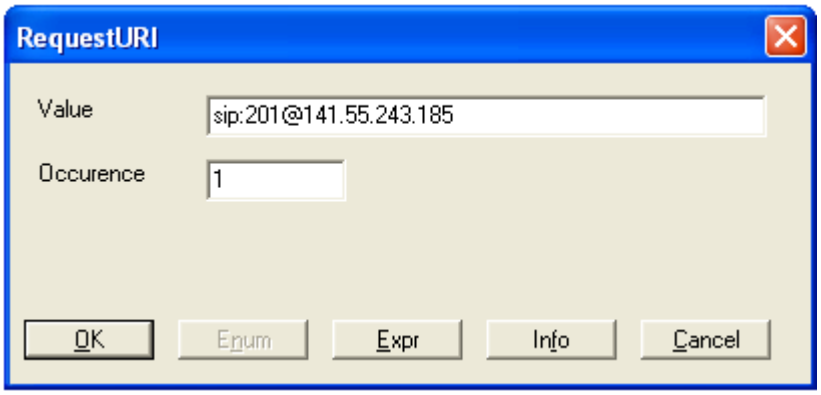

Abbildung 30: Request URI

Mit Cancel das Menü verlassen. Die Eingabe ist mit „Send;“ abzuschließen.

Schritt 2: Proxy Authentication Required

Der Asterisk-Server schickt die Nachricht „R407 Proxy Authentication Required“ zurück. Authentifizierung am Proxy ist erforderlich. Beginnen in der Menüleiste mit:

Script → Command → Wait for event [OK] → Message [OK] → Proxy Authentication Required (407) [OK]

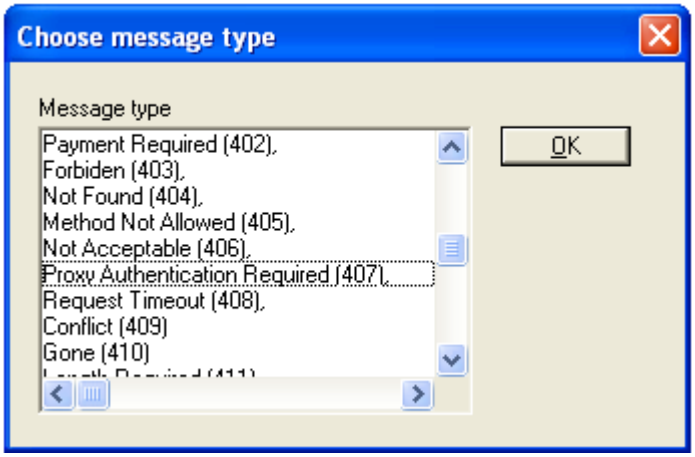

Abbildung 31: Messagetyp (407)

Schritt 3: Acknowledge

Der Asterisk-Server erwartet als nächstes eine Empfangsbestätigung. Die Nachricht hierfür ist „Acknowledge“ (ACK). Beginnen in der Menüleiste mit:

Script → Command → Put message into buffer [OK] → Modified message from pool [OK] → ACK [OK]

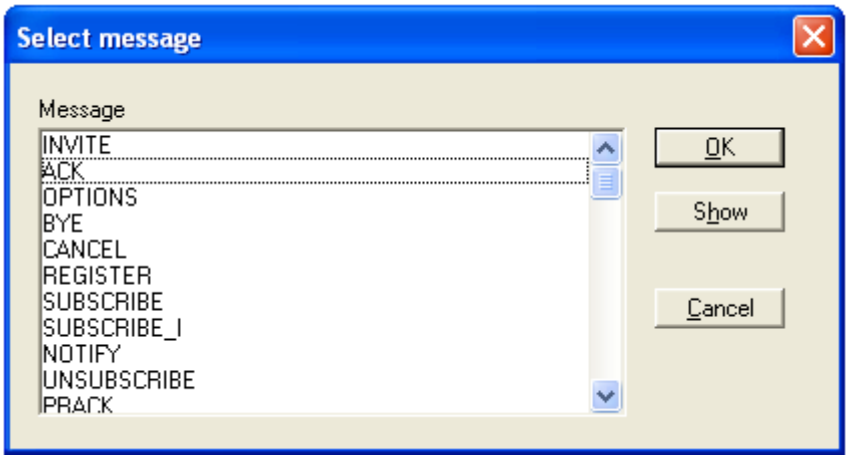

Abbildung 32: Nachricht auswählen

Durch Anklicken des Button Cancel ist das Menü zu verlassen.

Schritt 4: Invite wiederholen

Die Einladung zur Sitzung erfolgt zum zweiten Mal. Invite wird an den Asterisk-Server geschickt. Der Schritt 1 wird an dieser Stelle wiederholt.

Schritt 5: Trying und Ringing

Der Asterisk-Server schickt jetzt der Reihe nach die Nachrichten „R100 Trying“, „R180 Ringing“ und „R200 OK“. Die Vorgehensweise ist dem Schritt 2 zu entnehmen.

Schritt 6: Acknowledge

Durch das „Acknowledge“ am Ende wird dem Asterisk-Server der Verbindungsabbau bestätigt. An dieser Stelle kann Schritt 3 wiederholt werden.

Schritt 7: Eventhandler

Damit der Befehl Ringing nicht nur einmal ausgeführt wird und das Telefon mehrmals klingelt, ist es erforderlich, ein Handle Event in das Programm zu integrieren, welches sich auf die Nachricht Ringing bezieht. Durch den Eventhandler kann der normale Ablauf des Tests unterbrochen werden. Für diesen Test bedeutet das, dass durch die Nachricht Ringing (180) der Eventhandler aktiv wird und das Klingeln mehrfach als Nachricht empfangen wird, ohne

ein Error hervorzurufen. Das Handle Event ist nach dem Ende des Scripts einzufügen. Dazu beginnt man in der Menüleise mit:

Script → Command → Add event handler [OK] → Add local event handler [OK] → Message [OK] → Ringing (180) [OK]

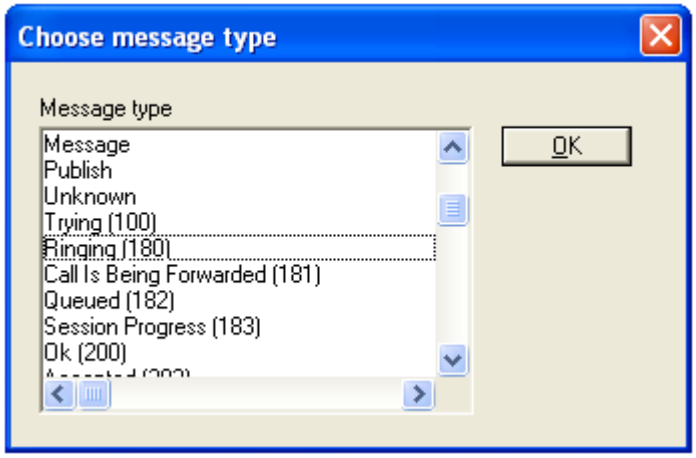

Abbildung 33: Messagetyp (180)

Durch Anklicken des Button Cancel ist das Menü zu verlassen.

Schritt 8: Speichern

Speichern des Testprogramms.

File → Save

Schritt 9: Kompilieren

Bevor der Test gestartet werden kann, muss das Testprogramm in der Control-Datei kompiliert werden. Dazu muss die Control-Datei geöffnet werden. In der Menüleiste:

File → Compile

Schritt 10: Testdurchführung

Der Test kann gestartet werden. Durch das Öffnen der Konfigurationsdatei wird der Test automatisch gestartet (Abbildung 34).

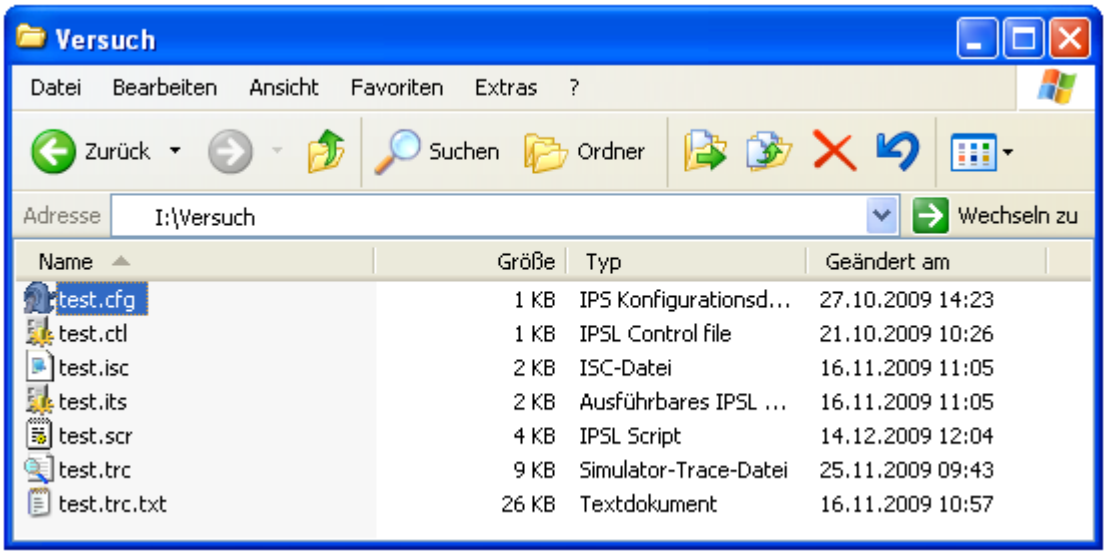

Abbildung 34: Teststart

Das IPS-Fenster mit dem Trace öffnet sich. Wenn das Telefon klingelt, muss der Hörer abgenommen werden, um diesen dann wieder aufzulegen. Dies ist erforderlich, um den Verbindungsabbau zu testen.

4.5 3. Versuchsaufgabe: Verbindung zu einem ISDN-Telefon

In der letzten Versuchsaufgabe soll getestet werden, ob ein erfolgreicher Verbindungsauf- und –abbau zu einem ISDN-Telefon durchgeführt werden kann. Die Verbindung soll über den Asterisk-Server hergestellt werden. Es soll dabei gezeigt werden, dass der Asterisk-Server verschiedene Endgeräte unterstützt. Der Ablauf ist der gleiche, wie in der zweiten Versuchsaufgabe. Lediglich die Rufnummer in der Invite-Nachricht ist zu ändern. Folgende Schritte sind zu gehen:

Schritt 1: Anpassen der ersten Invite-Nachricht

In der Nachricht Invite ist die Request-URI zu ändern. Die IP-Adresse des Asterisk-Servers bleibt dabei gleich. Es wird die Rufnummer des am Platz befindlichen ISDN-Telefons eingetragen. z.B.:

sip:2102@141.55.243.185

```
73    Put(Modify(Pool("INVITE", MSG_UNI_POOL))        /* |---- INVITE       --->| */
74          RequestURI := "sip:2102@141.55.243.185";
75       EndModify);
```

Abbildung 35: Rufnummer ISDN-Telefon

Schritt 2: Anpassen der zweiten Invite-Nachricht

Der erste Schritt ist bei der zweiten Invite-Nachricht zu wiederholen.

Schritt 3: Speichern

Speichern des Testprogramms.

File → Save

Schritt 4: Kompilieren

Bevor der Test gestartet werden kann, muss das Testprogramm in der Control-Datei kompiliert werden. Dazu muss die Control-Datei geöffnet werden. In der Menüleiste beginnen mit:

File → Compile

Schritt 5: Testdurchführung

Der Test kann gestartet werden. Durch das Öffnen der Konfigurationsdatei wird der Test automatisch gestartet. Das IPS-Fenster mit dem Trace öffnet sich. Wenn das Telefon klingelt, muss der Hörer abgenommen werden, um diesen dann wieder aufzulegen. Dies ist erforderlich, um den Verbindungsabbau zu testen.

4.6 Anhang zum Versuch

4.6.1 Nachrichten

Die Kommunikation bei SIP erfolgt über Nachrichten. Es werden Request- und Response-Nachrichten unterschieden. Einen Überblick über die auch im Versuch verwendeten Nachrichten, soll die folgende Zusammenstellung geben.

4.6.1.1 Request-Nachrichten

ACK	Empfangsbestätigung einer Nachricht oder Verbindungsvollendung
BYE	Einleitung des Abbaus einer bestehenden Verbindung
CANCEL	Abbruch der aktuellen Transaktion
INVITE	Einladung zu einer Sitzung
REGISTER	Übermittlung von Kontaktinformationen des Nutzers
SUBSCRIBE	Anmelden für Benachrichtigung bei Event

4.6.1.2 Response-Nachrichten

100	Trying (Versuch Verbindung herzustellen)
180	Ringing (die Gegenstelle klingelt)

200	OK
305	Use Proxy (Proxy wird benutzt)
401	Unauthorized (nicht autorisiert)
404	Not Found (Teilnehmer nicht gefunden)
407	Proxy Authentication Request (Authentifizierung am Proxy erforderlich)
486	Busy here (bestetzt)
503	Service Unavailable (Dienst nicht verfügbar
604	Does not exist anywhere (nicht existent)

4.6.2 Trace eines erfolgreichen Tests

```
TRC           : Trace opened: 2009-11-16_10:57:42           16 10:57:42.50
MAIN          : Independent Protocol Simulator (IPS)        16 10:57:42.50
              : IPS32, Build: 2009-10-09 (jk)
              : (c) Nokia Siemens Networks GmbH & Co. KG
              : Build time: 2009-10-09_13:56:38
              : Registered: 2009-10-13 (Koehn)
MAIN          : Reading IPS-Config:                         16 10:57:42.50
              : "C:\Dokumente und Einstellungen\telecom\Desktop\Diplomarbei
              : t\Versuch\test.cfg": OK
              : "c:\tools\ips.cfg": OK
MAIN          : RunMode: Running                            16 10:57:42.50
TIMER         : LoadCtrl: St=1000 Qu=1.000                  16 10:57:42.50
ITP           : Version: 2009-09-15                         16 10:57:42.50
SIP           : Version: 2009-09-29                         16 10:57:42.50
UDP           : Version: 2009-08-28                         16 10:57:42.50
UDP(LL)       : WinSock: Version 2.2                        16 10:57:42.50
ITP           : +-----------------------------------------+ 16 10:57:43.53
ITP           : | Control file:                           | 16 10:57:43.53
ITP           : |   test(sip01)                           | 16 10:57:43.53
ITP           : |     09.10.21_10:26:46  09.11.03_13:29:24 | 16 10:57:43.53
ITP           : | Scripts:                                | 16 10:57:43.53
ITP           : |   test(sip01)                           | 16 10:57:43.53
ITP           : |     09.11.03_13:29:20  09.11.03_13:29:24 | 16 10:57:43.53
ITP           : +-----------------------------------------+ 16 10:57:43.53
R000000-0 -0  : $
ó1 %s…”•□"qá!%aŽ“@Q"qá"%aŽ“@R"qá#%aŽ“%     16 10:57:43.67
R000000-0 -0  : $%@S"qá$%aŽ“@T"qá%%aŽ“@U"qá&%aŽ“@V"qá'%&     16 10:57:43.67
R000000-0 -0  : $&aŽ“@W"qá(%aŽ“@X"qá)%aŽ“@Y"qá*%aŽ“QP"q'     16 10:57:43.67
R000000-0 -0  : $'á+%aŽ“QQ"qá,%aŽ“QR"qá-%aŽ“QS"qá.%aŽ“Q%     16 10:57:43.67
R000000-0 -0  : $%T"qá/%aŽ“QU"qá`%aŽ“QV"qáa%aŽ“QW"qáb%a&     16 10:57:43.67
R000000-0 -0  : $&Ž“QX"qác%aŽ“QY"qád%aŽ“RP"qáe%aŽ“RQ"qá'     16 10:57:43.67
R000000-0 -0  : $'f%aŽ“RR"qág%aŽ“RS"qáh%aŽ“RT"qái%aŽ“RU%     16 10:57:43.67
R000000-0 -0  : $%"qáj%aŽ“RV"qák%aŽ“RW"qál%aŽ“RX"qám%aŽ&     16 10:57:43.67
R000000-0 -0  : $&“RY"qán%aŽ“SP"qáo%aŽ“SQ"qá0%s…”•□"qá1'     16 10:57:43.67
R000000-0 -0  : $'%aŽ“@Q"qá2%aŽ“@R"qá3%aŽ“@S"qá4%aŽ“@T"%     16 10:57:43.67
R000000-0 -0  : $%qá5%aŽ“@U"qá6%aŽ“@V"qá7%aŽ“@W"qá8%aŽ“&     16 10:57:43.67
R000000-0 -0  : $&@X"qá9%aŽ“@Y"qá:%aŽ“QP"qá;%aŽ“QQ"qá<%'     16 10:57:43.67
```

```
R000000-0 -0  : $'aŽ“QR"qá=%aŽ“QS"qá>%aŽ“QT"qá?%aŽ“QU"q%     16 10:57:43.67
R000000-0 -0  : $%áp%aŽ“QV"qáq%aŽ“QW"qár%aŽ“QX"qás%aŽ“Q&     16 10:57:43.67
R000000-0 -0  : $&Y"qát%aŽ“RP"qáu%aŽ“RQ"qáv%aŽ“RR"qáw%a'     16 10:57:43.67
R000000-0 -0  : $'Ž“RS"qáx%aŽ“RT"qáy%aŽ“RU"qáz%aŽ“RV"qá%     16 10:57:43.67
R000000-0 -0  : $%{%aŽ“RW"qá|%aŽ“RX"qá}%aŽ“RY"qá~%aŽ“SP&     16 10:57:43.67
R000000-0 -0  : $&"qá□%aŽ‘SQ"qà                             16 10:57:43.67
R000000-0 -0  : usr: |--.REGISTER------------------------>| 16 10:57:43.67
              : REGISTER sip:141.55.243.185 SIP/2.0
              : Contact: sip:141.55.243.185
              : To: sip:203@141.55.243.75
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 1 REGISTER
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.0
              : Content-Length: 0
              :
R000000-0 -0  : net: |<-.100 Trying-----------------------| 16 10:57:43.67
              : SIP/2.0 100 Trying
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.0
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 1 REGISTER
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:203@141.55.243.185>
              : Content-Length: 0
              :
R000000-0 -0  : net: |<-.401 Unauthorized------------------| 16 10:57:43.67
              : SIP/2.0 401 Unauthorized
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.0
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as0da439ec
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 1 REGISTER
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:203@141.55.243.185>
              : WWW-Authenticate: Digest realm="asterisk", nonce="154e56a5"
              : Content-Length: 0
              :
R000000-0 -0  : usr: |--.REGISTER------------------------>| 16 10:57:43.67
              : REGISTER sip:141.55.243.185 SIP/2.0
              : Contact: sip:141.55.243.185
              : To: sip:203@141.55.243.75
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 2 REGISTER
```

```
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.1
              : Content-Length: 0
              : Authorization: Digest username="203",realm="asterisk",nonce
              : ="154e56a5",uri="sip:141.55.243.185",response="7ee0e34641fb
              : 77f9a0dfb1e9922338fa"
              :
R000000-0 -0  : net: |<-.100 Trying-----------------------| 16 10:57:43.68
              : SIP/2.0 100 Trying
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.1
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 2 REGISTER
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:203@141.55.243.185>
              : Content-Length: 0
              :
R000000-0 -0  : net: |<-.200 OK---------------------------| 16 10:57:43.68
              : SIP/2.0 200 OK
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.1
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as0da439ec
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 2 REGISTER
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Expires: 120
              : Contact: <sip:141.55.243.185>;expires=120
              : Date: Mon, 16 Nov 2009 11:05:05 GMT
              : Content-Length: 0
              :
R000000-0 -0  : usr: |--.SUBSCRIBE----------------------->| 16 10:57:43.68
              : SUBSCRIBE sip:203@141.55.243.75 SIP/2.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 3 SUBSCRIBE
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75
              : Contact: sip:141.55.243.185
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.2
              : Content-Length: 0
              : Authorization: Digest username="203",realm="asterisk",nonce
              : ="154e56a5",uri="sip:203@141.55.243.75",response="7c39f288b
              : 3508d8f4e0eca3e8a824920"
              :
R000000-0 -0  : net: |<-.200 OK---------------------------| 16 10:57:43.68
```

```
              : SIP/2.0 200 OK
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.2
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 3 SUBSCRIBE
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:203@141.55.243.185>
              : Content-Length: 0
              :
R000000-0 -0  : usr: |--.INVITE-------------------------->| 16 10:57:43.68
              : INVITE sip:1404@141.55.243.185 SIP/2.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 4 INVITE
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Contact: sip:141.55.243.185
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.3
              : Content-Length: 0
              : Authorization: Digest username="203",realm="asterisk",nonce
              : ="154e56a5",uri="sip:1404@141.55.243.185",response="a1acb37
              : 4365a815522d881102af80c20"
              :
R000000-0 -0  : net: |<-.407 Proxy Authentication Require--| 16 10:57:43.68
              : SIP/2.0 407 Proxy Authentication Required
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.3
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 4 INVITE
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:1404@141.55.243.185>
              : Proxy-Authenticate: Digest realm="asterisk", nonce="530b173
              : e"
              : Content-Length: 0
              :
R000000-0 -0  : usr: |--.ACK------------------------------>| 16 10:57:43.68
              : ACK sip:1404@141.55.243.185 SIP/2.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 4 ACK
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.3
              : Content-Length: 0
```

```
              : Proxy-Authorization: Digest username="203",realm="asterisk"
              : ,nonce="154e56a5",uri="sip:1404@141.55.243.185",response="a
              : 1acb374365a815522d881102af80c20"
              :
R000000-0 -0  : usr: |--.INVITE-------------------------->| 16 10:57:43.68
              : INVITE sip:1404@141.55.243.185 SIP/2.0
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 5 INVITE
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Contact: sip:141.55.243.185
              : Max-Forwards: 70
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.4
              : Content-Length: 0
              : Proxy-Authorization: Digest username="203",realm="asterisk"
              : ,nonce="530b173e",uri="sip:1404@141.55.243.185",response="4
              : cb6e77b47d1ec43cef28e599e722e5a"
              :
R000000-0 -0  : net: |<-.100 Trying-----------------------| 16 10:57:43.70
              : SIP/2.0 100 Trying
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.4
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 5 INVITE
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:1404@141.55.243.185>
              : Content-Length: 0
              :
R000000-0 -0  : net: |<-.180 Ringing----------------------| 16 10:57:43.71
              : SIP/2.0 180 Ringing
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.4
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
              : CSeq: 5 INVITE
              : User-Agent: Asterisk PBX
              : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
              : Contact: <sip:1404@141.55.243.185>
              : Content-Length: 0
              :
R000000-0 -0  : net: |<-.180 Ringing----------------------| 16 10:57:44.10
              : SIP/2.0 180 Ringing
              : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
              : 3568.4
              : From: sip:203@141.55.243.75;tag=3568.12080390.0
              : To: sip:203@141.55.243.75;tag=as72bb0b89
              : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
```

```
             : CSeq: 5 INVITE
             : User-Agent: Asterisk PBX
             : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
             : Contact: <sip:1404@141.55.243.185>
             : Content-Length: 0
             :
R000000-0 -0 : net: |<-.200 OK--------------------------| 16 10:57:46.37
             : SIP/2.0 200 OK
             : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
             : 3568.4
             : From: sip:203@141.55.243.75;tag=3568.12080390.0
             : To: sip:203@141.55.243.75;tag=as72bb0b89
             : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
             : CSeq: 5 INVITE
             : User-Agent: Asterisk PBX
             : Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER
             : Contact: <sip:1404@141.55.243.185>
             : Content-Type: application/sdp
             : Content-Length: 242
             :
             : v=0
             : o=root 2520 2520 IN IP4 141.55.243.185
             : s=session
             : c=IN IP4 141.55.243.185
             : t=0 0
             : m=audio 13464 RTP/AVP 0 8 101
             : a=rtpmap:0 PCMU/8000
             : a=rtpmap:8 PCMA/8000
             : a=rtpmap:101 telephone-event/8000
             : a=fmtp:101 0-16
             : a=silenceSupp:off - - - -
R000000-0 -0 : usr: |--.ACK----------------------------->| 16 10:57:46.37
             : ACK sip:1404@141.55.243.185 SIP/2.0
             : Call-ID: 00000000-00000DF0-00B85506-0@141.55.243.75
             : CSeq: 5 ACK
             : From: sip:203@141.55.243.75;tag=3568.12080390.0
             : To: sip:203@141.55.243.75;tag=as72bb0b89
             : Max-Forwards: 70
             : Via: SIP/2.0/UDP 141.55.243.75:2321;branch=z9hG4bK12080390.
             : 3568.5
             : Content-Length: 0
             : Proxy-Authorization: Digest username="203",realm="asterisk"
             : ,nonce="530b173e",uri="sip:1404@141.55.243.185",response="4
             : cb6e77b47d1ec43cef28e599e722e5a"
             :
R000000-0 -0 : $‘(”…“”N“ƒ’ë0*YK                          16 10:57:46.37
MAIN         : Shutdown started...                        16 10:57:48.68
MAIN         : Waiting for layers to finish...            16 10:57:48.68
R000000-0 -0 : -------------- BEGIN of statistic  1 (A)   16 10:57:48.68
R000000-0 -0 : Statistic on A-side;           Seiz/Fail   16 10:57:48.68
R000000-0 -0 : ---------------------------------------    16 10:57:48.68
R000000-0 -0 : A001=                                1     16 10:57:48.68
```

```
R000000-0 -0  : B001=                                    0     16 10:57:48.68
R000000-0 -0  : ---------------------------------------     16 10:57:48.68
R000000-0 -0  : NUMBER OF CALLS              :         1     16 10:57:48.68
R000000-0 -0  : NUMBER OF NOT FAILED CALLS   :         1     16 10:57:48.68
R000000-0 -0  : NUMBER OF FAILED CALLS       :         0     16 10:57:48.68
R000000-0 -0  : FAILURE RATE                 :    0.000%     16 10:57:48.68
R000000-0 -0  : NUMBER OF PENDING CALLS      :         0     16 10:57:48.68
R000000-0 -0  : MAX SIMULTANEOUS CALLS       :         1     16 10:57:48.68
R000000-0 -0  : NUMBER OF NOT FAILED CALLS: 1                16 10:57:48.68
R000000-0 -0  : --------------  END  of statistic  1         16 10:57:48.68
R000000-0 -0  : -------------- BEGIN of statistic  2 (B)     16 10:57:48.68
R000000-0 -0  : Statistic on B-side;           Seiz/Fail     16 10:57:48.68
R000000-0 -0  : ---------------------------------------     16 10:57:48.68
R000000-0 -0  : ---------------------------------------     16 10:57:48.68
R000000-0 -0  : NUMBER OF CALLS              :         0     16 10:57:48.68
R000000-0 -0  : NUMBER OF NOT FAILED CALLS   :         0     16 10:57:48.68
R000000-0 -0  : NUMBER OF FAILED CALLS       :         0     16 10:57:48.68
R000000-0 -0  : FAILURE RATE                 :    0.000%     16 10:57:48.68
R000000-0 -0  : NUMBER OF PENDING CALLS      :         0     16 10:57:48.68
R000000-0 -0  : MAX SIMULTANEOUS CALLS       :         0     16 10:57:48.68
R000000-0 -0  : NUMBER OF NOT FAILED CALLS: 0                16 10:57:48.68
R000000-0 -0  : --------------  END  of statistic  2         16 10:57:48.68
R000000-0 -0  : SEZ:00000000001 ERR:00000000000   0.000%     16 10:57:48.68
R000000-0 -0  : O  :00000000001 ERR:00000000000   0.000%     16 10:57:48.68
R000000-0 -0  : I  :00000000000 ERR:00000000000   0.000%     16 10:57:48.68
UDP           : -------------- BEGIN of statistic    (T)     16 10:57:48.68
UDP           : Link D      Bytes   Msg's  Bytes/s  Msg's/s  16 10:57:48.68
UDP           :    0 O       3119       7        0        0  16 10:57:48.68
UDP           :    0 I       4243      10        0        0  16 10:57:48.68
UDP           : --------------  END of statistic     (T)     16 10:57:48.68
ITP           : +----------------------------------------+ 16 10:57:48.68
ITP           : | Control file:                          | 16 10:57:48.68
ITP           : |   test(sip01)                          | 16 10:57:48.68
ITP           : |     09.10.21_10:26:46  09.11.03_13:29:24 | 16 10:57:48.68
ITP           : | Scripts:                               | 16 10:57:48.68
ITP           : |   test(sip01)                          | 16 10:57:48.68
ITP           : |     09.11.03_13:29:20  09.11.03_13:29:24 | 16 10:57:48.68
ITP           : +----------------------------------------+ 16 10:57:48.68
MAIN          : Shutdown complete!                         16 10:57:48.70
MAIN          : ITP-Mem: +24264, -24264                    16 10:57:48.71
MAIN          : STAT-Mem: +90440, -90440                   16 10:57:48.71
TRC       : Trace closed: 2009-11-16_10:57:48        16 10:57:48.71
```

5 Zusammenfassung und Schlussfolgerungen

Kommunikationssysteme für die allgemeine oder kundenspezifische Anwendung müssen vor ihrem praktischen Gebrauch getestet werden. Damit soll geprüft werden, ob die gestellten spezifischen Anforderungen von den Kommunikationssystemen auch sicher erfüllt und Störungen weitestgehend ausgeschlossen werden können.
Ausgehend davon wurde mit der vorliegenden Diplomarbeit ein Praktikumsversuch für Studenten der Informations- und Elektrotechnik erarbeitet, mit dem ein Test durchgeführt werden kann. Der Erarbeitung des Praktikumsversuchs liegen Ausführungen zu den theoretischen Grundlagen der Softwareentwicklung und deren Testung sowie ein Eigenversuch (Funktionaltest) mit den Schwerpunkten Testaufbau, Testvorbereitung und Testdurchführung zugrunde. Festgelegte Funktionalitäten eines Systems wurden damit geprüft. Dem Eigenversuch folgt die Beschreibung des Praktikumsversuches.
Schlussfolgernd ist als Ergebnis festzustellen:

- Es liegt eine umfangreiche Versuchsanleitung für einen Praktikumsversuch für Studenten zum Thema Testen vor.
- Funktionalitäten wurden mit dem Eigenversuch erfolgreich getestet.
- Der Praktikumsversuch kann durch zusätzliche Tests erweitert werden, z.B. mit dem Performancetest.
- Die Arbeit mit der IPS-Software erwies sich anfänglich als etwas schwierig, da noch keine Erfahrungen mit deren Anwendung vorlagen.

6 Anlagen

6.1 Script-Datei des Funktionaltests Kapitel 3

```
SslVersion 1.1;

Script Client;

Activated Locally;

Var
  name           : String;  //Variable "name" vom Typ String
  tiuAddr        : Integer;   //Variable "tiuAddr" vom Typ Integer (Tone Interface Unit)
  myRTPAddress   : String;    //Variable "myRTPAddress" vom Typ String
  remoteRTPPort  : String;  //Variable "remoteRTPPort" vom Typ String
  R200_SDPString : String;  //Variable "R200_SDPSring" vom Typ String


BeginScript

    /* RTP */

  // Choose a name for the endpoint
  name := "Res" & ResourceNumber();      //Namen für den Endpunkt auswählen   Name: Res0
  // Create an endpoint with
  //    IP address = local host address
  //    RTP port   = 49972
  //    RTCP port  = 49973 (default 10000+1)
  SendInternalMessage(InvokeData, "TIU", "CrEP:RTPOnHost:" & name & ",,49972");


  // Wait for the notification that endpoint is created
  StartTimer(t_Tiu, 5*SEC); //Start Timer mit 5 Sekunden Laufzeit (Tone Intervace Unit)
  WaitFor InternalMessage(Data, ParseString(InternalMessageData(), ",", 1) = "NTFY:CrEP");
//Warten auf Nachricht
  StopTimer(t_Tiu);   //Stopt Timer

  tiuAddr := InternalMessageOrigination();
  InternalCancelOnError(tiuAddr, On);

  // in case the ports are determined by the IPSLTiu
  // maybe they are needed here to send them to the other
  // side e.g. via a session description in SIP
  myRTPAddress := ParseString(InternalMessageData(), ",", 2);  //erfassen der RTP-Adresse über
eine Variable
  PRINT("My RTP/RTCP address: ip_addr=" & ParseString(InternalMessageData(), ",", 2) &
//sepaieren IP-Adresse
        ", RTP_port=" & ParseString(InternalMessageData(), ",", 3) &
        ", RTCP_port=" & ParseString(InternalMessageData(), ",", 4) );
```

```
  Put(Modify(Pool("INVITE", MSG_UNI_POOL))       /* |---- INVITE      --->| */ //Einleitung
einer Verbindung
          RequestURI := "sip:141.55.241.246";      //IP B-Teilnehmer

          //Insert c_P_Request;
          Insert c_P_RequestHeader.c_P_From;
          From := "Hello World!";     //Info über SIP-Adresse des A-Teilnehmes
          Insert c_P_Request.c_P_Body;
          Insert c_P_Body.c_P_BodyText;
          BodyText := "v=0" & Chr(13) &     //v= Protokollversion
                      "o=- 1234567890 7894561230 IN IP4 141.55.243.75" & Chr(13) & //o= Sessi-
on-Ursprung und IP
                      "s=IPS is the best"   &  Chr(13) &  //s= Session-Name
                      "m=audio 49972 RTP/AVP 8 0" &  Chr(13) &   //m= Medienname und Übertra-
gungsadresse
                      "c=IN IP4 141.55.243.75" &  Chr(13) &
                      "t=0 0" &   Chr(13) &   //t= Zeitgrenze (Anfang, Ende)
                      "a=rtpmap:8 PCMA/8000" &  Chr(13) &
                      "a=rtpmap:0 PCMU/8000";

      EndModify);
  Send;


  StartTimer(tclient, 15*SEC);           //startet den Timer "tclient" 15 Sekunden
  WaitFor Message(R100_TRYING);                   /* |<--- R100_TRYING ----| */  //Nachricht
Trying
  WaitFor Message(R180_RINGING);                /* |<--- R180_RINGING ----| */ //warten auf
die Nachricht R180_Ringing
  WaitFor Message(R200_OK);                          /* |<--- R200_OK     ----| */  //warten
auf die Nachricht R200_OK
  StopTimer(tclient);                 //stopt den Timer

  R200_SDPString := GetPar(BodyText); //liefern den Parameter "Body" aus der letzten Nachricht
R200_SDPString
  PRINT(R200_SDPString);   //Parameterausgabe"Body"des SessionDescriptionProtocolls im Trace
viewer

  R200_SDPString := ParseString(R200_SDPString, "m=audio ", 2);  //separieren Mediennamens und
Adresse für Transport
  PRINT(R200_SDPString);       //Ausgabe Medienname und Adresse (Port) im Trace viewer


  remoteRTPPort := ParseString(R200_SDPString, " ", 1);   //separieren des RTPPorts
  PRINT(remoteRTPPort);      //Ausgabe des RTPPots im Trace viewer


  SendInternalMessage(Data, tiuAddr, "SetEP:141.55.241.246," & remoteRTPPort);   //senden der
IP A-Tln und des RTP-Pots
```

```
  // Switch on an 1 kHz oscillator
  SendInternalMessage(Data, tiuAddr, "Snd1kHz"); //senden eines 1kHz-Tones zur akustischen
Kontrolle

  StartTimer(tclient, 30000);   //startete den  Timer "tclient"
  WaitFor Message(R200_OK);                      /* |<--- R200_OK     ----| */ //warten auf
Nachricht "R200_OK"
  StopTimer(tclient);      //stopt den Timer

    // Finally delete the session
  SendInternalMessage(Data, tiuAddr, "DelEP:BYE");

EndScript;      //Script Ende

HandleEvent Message(R180_RINGING)    //Ausgabe "Ringing 180 RECIVED" im Trace viewer B-Tln
klingelt
  PRINT(" >>>> RINGING 180 RECIVED");
EndHandleEvent;
```

6.2 Quelltext der Scrip-Datei des Praktikumsversuches

```
/**TMS*****************************************************/
/*                                                       */
/*TEST-CASE    : test.scr                                */
/*                                                       */
/*PURPOSE      : SIP Versuch                             */
/*                                                       */
/*DEPARTMENT   : Kommunikationstechnik                   */
/*AUTHOR       : telecom.tc-060404-04                    */
/*                                                       */
/*******************************************************/
/*<<<DESCRIPTION>>>*/
/*<<<SCRIPT>>>*/

/* Generated with SIP01 */

IpslVersion 2.0;

Script test;

Activated Locally;


BeginScript

  DIGEST_User     := "203";     //Nutzername
  DIGEST_Password := "abc123";  //Password des Nutzers
```

```
//Ab hier Programmtext!!!

  Put(Modify(Pool("REGISTER", MSG_UNI_POOL))     /* |---- REGISTER    --->| */
         RequestURI := "sip:141.55.243.185";
         Insert c_P_RequestHeader.c_P_Contact;
         Contact := "sip:141.55.243.185";
         Insert c_P_RequestHeader.c_P_To;
         To := "sip:203@141.55.243.75";
         Insert c_P_RequestHeader.c_P_From;
         From := "sip:203@141.55.243.75";
      EndModify);

   Send;

   WaitFor Message(R100_TRYING);                 /* |<--- R100_TRYING ----| */
   WaitFor Message(R401_UNAUTHORIZED);           /* |<--- R401_UNAUTHORIZED-|*/

  Put(Modify(Pool("REGISTER", MSG_UNI_POOL))     /* |---- REGISTER    --->| */
        RequestURI := "sip:141.55.243.185";
        Insert c_P_RequestHeader.c_P_Contact;
        Contact := "sip:141.55.243.185";
        Insert c_P_RequestHeader.c_P_To;
        To := "sip:203@141.55.243.75";
        Insert c_P_RequestHeader.c_P_From;
        From := "sip:203@141.55.243.75";
      EndModify);

   Send;

   WaitFor Message(R100_TRYING);                 /* |<--- R100_TRYING ----| */
   WaitFor Message(R200_OK);                     /* |<--- R200_OK     ----| */

     Put(Modify(Pool("SUBSCRIBE_I", MSG_UNI_POOL))/* |---- SUBSCRIBE_I -->| */
          RequestURI := "sip:203@141.55.243.75";
      EndModify);

   Send;


   WaitFor Message(R200_OK);                     /* |<--- R200_OK     ----| */

//bis hier Registrierung
//ab hier Verbindungsaufbau

   Put(Modify(Pool("INVITE", MSG_UNI_POOL))      /* |---- INVITE      --->| */
         RequestURI := "sip:1404@141.55.243.185";
      EndModify);

   Send;

   WaitFor Message(R407_PROXY_AUTH_REQ);      /* |<--- R407_PROXY_AUTH_REQ -| */
```

```
   Put(Modify(Pool("ACK", MSG_UNI_POOL))          /* |---- ACK         --->| */
      EndModify);

   Send;

   Put(Modify(Pool("INVITE", MSG_UNI_POOL))       /* |---- INVITE      --->| */
         RequestURI := "sip:1404@141.55.243.185";
      EndModify);

   Send;

   WaitFor Message(R100_TRYING);                  /* |<--- R100_TRYING ----| */
   WaitFor Message(R180_RINGING);                 /* |<--- R180_RINGING ----| */
   WaitFor Message(R200_OK);                      /* |<--- R200_OK     ----| */

   Put(Modify(Pool("ACK", MSG_UNI_POOL))          /* |---- ACK         --->| */
      EndModify);

   Send;

//Ende Programmtext

EndScript;

//ab hier event handler

   HandleEvent Message(R180_RINGING)
   EndHandleEvent;
```

Literaturverzeichnis

[Balzert, 1992]

Balzert, Helmut: Die Entwicklung von Software-Systmen. Prinzipien, Methoden, Sprachen, Werkzeuge -2. unveränderte Aufl.- Mannheim: Bibliographisches Institut, 1992, ISBN 3-411-01618-3

[Bergmann, 2003]

Bergmann, Friedhelm; Gerhardt, Hans-Joachim; Frohberg, Wolfgang: Taschenbuch der Telekommunikation. -2. Aufl.- Leipzig: Fachbuchverlag Leipzig im Carl Hanser Verlag, 2003, ISBN 3-446-21750-9

[Franz, 2007]

Franz, Klaus: Handbuch zum Testen von Web-Applikationen. Berlin Heidelberg: Springerverlag, 2007, ISBN 978-3-540-24539-1

[Liggesmeyer, 2009]

Liggesmeyer, Peter: Software-Qualität. Testen, Analysieren und Verifizieren von Software -2. Aufl.- Heidelberg: Spektrum Akademischer Verlag, 2009, ISBN 978-3-8274-2056-5

[Petrasch, 2005]

Petrasch, Roland; Höhn, Reinhard; Höppner, Stephan; Wetzel, Herbert; Wirms, Manuela: Entscheidungsfall Vorgehensmodell. -1. Aufl.- Achen: Shaker Verlag, 2005, ISBN 3-8322-3914-6

[Rabe, 2008]

Rabe, Markus; Spickermann, Sven; Wenzel, Sigrid: Verifikation Validierung für die Simulation in Produktion und Logistick. Berlin Heidelberg: Springerverlag 2008, ISBN 978-3-540-35281-5

[Spillner, 2005]

Spillner, Andreas; Linz, Tilo: Basiswissen Softwaretest. -3.Aufl.- Heidelberg: Dpunkt Verlag, 2005, ISBN 3-89864-358-1

[Nokia-Siemens-Networks, 2009]
IPS-Onlinehandbuch: interne Firmendokumentation, Nokia-Siemens-Networks, 2009

[IETF, 2009]
IETF: RFC 3261, http://tools.ietf.org/html/rfc3261, verfügbar am 16. 06. 2009

[IETF, 2010]
IETF: RFC 4566, http://tools.ietf.org/html/rfc4566, verfügbar am 20. 01. 2010

[infforum, 2009]
infforum: agile: http://www.infforum.de/themen/anwendungsentwicklung/thema_SE_agile_software-entwicklung.htm, verfügbar am 03.12.2009

[ITWissen, 2009]
IT-Wissen: Konformancetest, http://www.itwissen.info/definition/lexikon/Konformitaetstest-CTconformancetesting.html, verfügbar am 20.11.2009

[Netzwerkguide, 2009]
Netzwerkguide: Voice over IP, http://www.netzwerkguide.com/voice-over-ip, verfügbar am 15.06.2009

[qse, 2009]
qse: Testfälle, http://qse.ifs.tuwien.ac.at/courses/scriptum/download/09P_Funkt_wid_20040204.pdf, verfügbar am 09.12.2009

[software-kompetenz, 2010]
software-kompetenz: TTCN-3, http://www.software-kompetenz.de/?26812, verfügbar am 21.01.2010

[Verifysoft, 2010]
Verifysoft: Coveragetest, http://www.verifysoft.com/de_ctcpp.html, verfügbar am 20.01.2010

[VoIP-info,2010]

VoIP-Information: SIP-Header, http://www.voip-information.de/sip/header-format.php, verfügbar am 02.02.2010

[Vorlesung, 2009]

Vorlesung Professur Telekommunikation, http://www.htwm.de/~telecom/, 2009

[Wikipedia, 2009]

Wikipedia: SIP, http://de.wikipedia.org/wiki/Session_Initiation_Protocol, verfügbar am 29.09.2009

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Osterburg, 14 Februar 2010